Reproduction Information

Cost uncertainties and ecological impacts drive tradeoffs between electrical system decarbonization pathways in New England, U.S.A.

Amir M. Gazar[1,2], Chloe Jackson[3], Georgia Mavrommati[3], Rich B. Howarth[4], and Ryan S.D. Calder[1,2,5,*]

[1]Department of Population Health Sciences, Virginia Tech, Blacksburg, VA, 24061, USA
[2]Global Change Center, Virginia Tech, Blacksburg, VA, 24061, USA
[3]School for the Environment, University of Massachusetts Boston, Boston, MA 02125, USA
[4]Environmental Program, Dartmouth College, Hanover, NH, 03755, USA
[5]Department of Civil and Environmental Engineering, Duke University, Durham, NC, 27708, USA
[*]Corresponding author: `rsdc@vt.edu`

Contents

1 Configuration and Setup

This document presents a comprehensive guide outlining the sequential process required to execute and replicate the capacity expansion model and total cost calculation presented in our article. Additionally, we provide a conceptual overview of code functionality in Section 2.2.

1.1 Hardware and Operating System

We utilized a MacBook Pro with an Apple M1 Pro chip, featuring an 8-core CPU and 16 GB of memory. The startup disk is the Macintosh HD. The system operates on macOS 14.5 `Sonoma`. To run the generation expansion model script, we utilized Virginia Tech's Advanced Research Computing (ARC) resources. We used the following specifications for this source for each simulation.

```
#SBATCH --partition=normal_q
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=50
#SBATCH --cpus-per-task=1
#SBATCH --time=24:00:00
```

1.2 Software Versions and Dependencies

We used R version 4.4.2 `(2024-10-31)` for the x86_64-apple-darwin20 platform. Furthermore, we used the RStudio environment, version 2024.09.1 Build 394 to execute the code. Table 1 shows theackages and versions utilized in this code.

Table 1: Package Version

Package	Version
data.table	1.14.8
lubridate	1.9.3
httr	1.4.6
jsonlite	1.8.7
htmltools	0.5.5
ggplot2	3.4.2
scales	1.2.1
zoo	1.8.12
gridExtra	2.3
dplyr	1.1.2

1.3 Install Time

Installation of RStudio and R depends on the specific hardware and operating system in use. This also holds true for the packages employed within the code. However, the packages used in this script typically take a few minutes to install.

1.4 Run Time

The run time to execute the generation expansion model script in ARC at Virginia Tech is as follows:

- Full demand curve (9,495 days): 15 minutes for 1 simulation for each decarbonization pathway.

Depending on the size of the data set and the hardware configuration, this duration may vary. For all other scripts, the run time was a few minutes on our computer.

2 Conceptual Overview and Code Availability

2.1 Code availability

The codes for this study are available on the GitHub repository via this link.

2.2 Conceptual Overview

The framework integrates key components across three stages: (1) Input Variables, which include decarbonization pathways (e.g., new offshore wind, transmission lines to/from Quebec, and new small modular reactors) and external data sources (e.g., ISO-NE demand, EIA and EPA datasets, and IEA resources); (2) the Generation Expansion Model, which computes demand curves, clean and fossil fuel generation, imports, energy storage, and unmet demand at the Regional Transmission Organization (RTO) level; and (3) Total Costs, which are determined through cost modules covering CAPEX, FOM, VOM, emissions, fuel, imports, and unmet demand penalties to estimate total social costs.

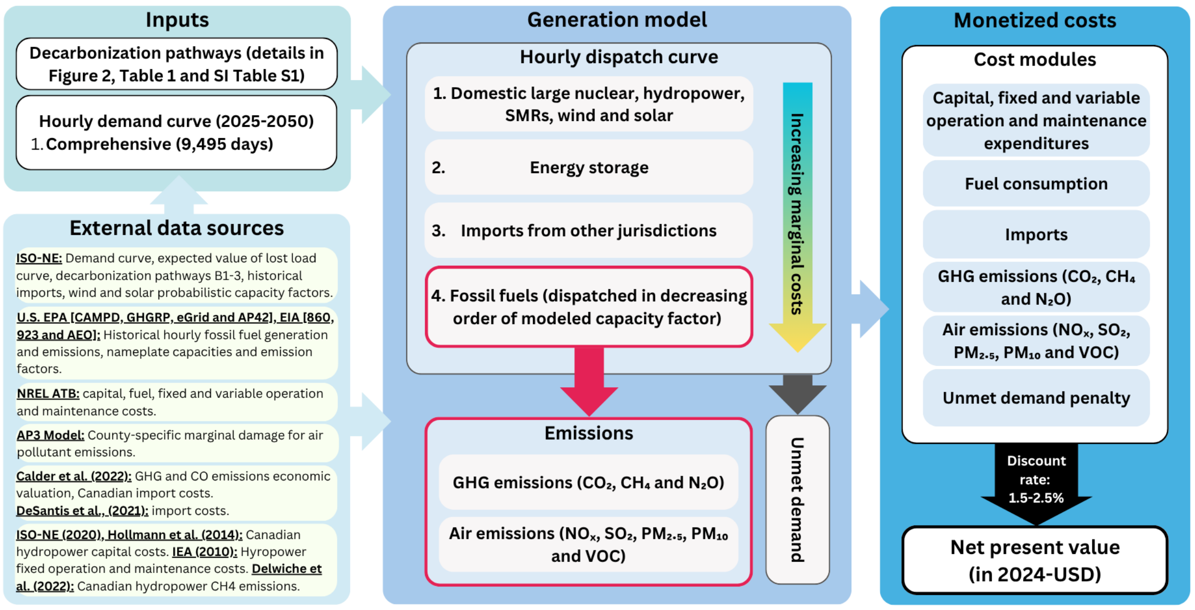
Inputs
Decarbonization pathways (details in Figure 2, Table 1 and SI Table S1)
Hourly demand curve (2025-2050)
1. Comprehensive (9,495 days)
External data sources
ISO-NE: Demand curve, expected value of lost load curve, decarbonization pathways B1-3, historical imports, wind and solar probabilistic capacity factors.
U.S. EPA [CAMPD, GHGRP, eGrid and AP42], EIA [860, 923 and AEO]: Historical hourly fossil fuel generation and emissions, nameplate capacities and emission factors.
NREL ATB: capital, fuel, fixed and variable operation and maintenance costs.
AP3 Model: County-specific marginal damage for air pollutant emissions.
Calder et al. (2022): GHG and CO emissions economic valuation, Canadian import costs.
DeSantis et al., (2021): import costs.
ISO-NE (2020), Hollmann et al. (2014): Canadian hydropower capital costs. IEA (2010): Hyropower fixed operation and maintenance costs. Delwiche et al. (2022): Canadian hydropower CH4 emissions.
Generation model
Hourly dispatch curve
1. Domestic large nuclear, hydropower, SMRs, wind and solar
2. Energy storage
3. Imports from other jurisdictions
4. Fossil fuels (dispatched in decreasing order of modeled capacity factor)
Increasing marginal costs
Emissions
GHG emissions (CO_2, CH_4 and N_2O)
Air emissions (NO_x, SO_2, $PM_{2.5}$, PM_{10} and VOC)
Unmet demand
Monetized costs
Cost modules
Capital, fixed and variable operation and maintenance expenditures
Fuel consumption
Imports
GHG emissions (CO_2, CH_4 and N_2O)
Air emissions (NO_x, SO_2, $PM_{2.5}$, PM_{10} and VOC)
Unmet demand penalty
Discount rate: 1.5-2.5%
Net present value (in 2024-USD)

Figure 1: Conceptual overview of the modeling framework.

3 Decarbonization Pathways

3.1 Decarbonization Pathways Data Processing

The decarbonization pathways are analyzed using a custom R script, `Hourly_Installed_Capacity.R`, which processes data from an Excel file. The key steps are as follows:

1. **Loading Libraries**: The script utilizes the `readxl`, `data.table`, and `lubridate` libraries for reading Excel files, managing data efficiently, and handling date-time information, respectively.

2. **Reading Decarbonization Data**:
   - The Excel file, `Decarbonization_Pathways.xlsx`, contains multiple sheets, each representing a decarbonization pathway.
   - All sheets are read and combined into a unified dataset.

3. **Adding Metadata**: A new column `Pathway` is added to the data from each sheet, identifying the respective pathway.

4. **Combining Data**: The data from all sheets are combined into a single `data.table` for further analysis.

5. **Extracting Year Range**: The script identifies the range of years in the dataset (`start_year` and `end_year`) for contextualizing analysis.

Important Functions and Steps:

- Loading Excel sheet names:

Listing 1: Loading Excel Sheet Names

```
    sheet_names <- excel_sheets(file_path)
```

- Adding pathway metadata to each sheet:

Listing 2: Adding Pathway Metadata

```
    sheet_data[, Pathway := sheet]
```

- Combining all data:

Listing 3: Combining All Sheets

```
    combined_data <- rbindlist(data_tables, fill = TRUE)
```

- Extracting year range:

Listing 4: Extracting Year Range

```
    years <- unique(as.numeric(combined_data$Year))
    start_year <- min(years, na.rm = TRUE)
    end_year <- max(years, na.rm = TRUE)
```

This script prepares the decarbonization pathways data for subsequent analysis, ensuring all relevant information is integrated and organized.

4 Generation Expansion Model

4.1 Wind and Solar Hourly Capacity Factors

4.1.1 Overview

The `Wind_Solar_Hourly_CF.R` script processes hourly wind and solar generation data to calculate capacity factors. The script performs several preprocessing steps to standardize the data and align it with the time zones and calendar conventions used in the model.

4.1.2 Detailed Explanation

1. **Loading Data**: Hourly generation data is read from CSV files, and the required columns are selected for analysis.

2. **Time Zone Adjustment**: Timestamps are converted from UTC to Eastern Standard Time (EST), aligning with local conditions.

3. **Handling Leap Years**: Leap year adjustments ensure the data is consistent, avoiding errors in models that depend on the length of the year.

4.1.3 Key Code

Listing 5: Adjusting Timestamps and Handling Leap Years

```
data_processing_func <- function(file, columns_to_keep) {
  data <- readr::read_csv(file, show_col_types = FALSE)
  data <- data %>% dplyr::select(all_of(columns_to_keep))
```

```r
  data$Date <- as.POSIXct(data$Date, format = "%m/%d/%Y %I:%M:%S %p", tz = "UTC")

  # Convert UTC to EST
  data$Date <- data$Date - hours(5)

  # Add DayLabel and Hour columns
  data <- data %>%
    mutate(DayLabel = yday(Date), Hour = hour(Date) + 1)

  # Adjust DayLabel for non-leap years
  if (!is_leap_year(year(data$Date[365]))) {
    data <- data %>% mutate(DayLabel = ifelse(DayLabel == 366, 365, DayLabel))
  }
}
```

4.2 Small Modular Reactors (SMR)

4.2.1 Overview

The SMR_CF.R script models the operational characteristics of small modular reactors. These include:

- Capacity factor (90%)
- Ramp rate and minimum power output
- Core fuel characteristics, including enrichment and burnup

4.2.2 Key Code

Listing 6: Defining SMR Specifications

```r
data <- data.frame(
  Label = "SMR300",
  Category = "SMR",
  Nameplate_Capacity_MW = 300,
  CF = 0.9,
  Ramp = 1,
  Minimum_Power_Output_MW = 60,
  Enrichment = "4.95%",
  Fuel = "UO2",
  Reactor_Lifetime = 60
)
```

4.3 Fossil Fuels: Facility Data and Emissions

4.3.1 Fossil Fuel Facility Data

The Fossil_Fuels_Facilities_Data.R script processes historical fossil fuel facility data for New England. This data is critical for understanding the existing capabilities and limitations of the regional fossil fuel infrastructure.

Detailed Explanation:

1. **Loading Data**: Facility data for all U.S. states is loaded. The dataset includes information on facility location, capacity, operating status, and ramp rates.

2. **Filtering for Active Facilities**: Facilities are filtered to include only those located in New England and those that are operational (i.e., not retired).

3. **Ramp Rate Conversion**: Ramping times (e.g., ”10M”, ”1H”) are converted into numeric values representing hours. This standardization ensures compatibility with the model.

Key Code Excerpts:

Listing 7: Ramp Rate Conversion Function

```r
convert_ramp_to_numeric <- function(ramp) {
  if (is.na(ramp)) {
    return(24) # Default to 24 hours for missing values
  } else if (ramp == "10M") {
    return(0.1666667)  # 10 minutes
  } else if (ramp == "1H") {
    return(1)  # 1 hour
  } else if (ramp == "12H") {
    return(12)  # 12 hours
  } else if (ramp == "OVER") {
    return(24)  # Default for 'OVER'
  } else {
    return(24)
  }
}
```

Listing 8: Filtering Facilities for New England

```r
# Filter data for active facilities in New England
new_england_states <- c("CT", "ME", "MA", "NH", "RI", "VT")
facilities_data_NE <- facilities_data[State %in% new_england_states]
facilities_data_NE <- facilities_data_NE[!grepl("retired", Operating_Status, ignore.case = ...
    TRUE)]
```

4.3.2 Fossil Fuel Generation Emissions

The `Fossil_Fuels_Gen_Emissions.R` script compiles hourly emissions data for fossil fuel facilities in New England. Emissions data is crucial for estimating environmental costs and compliance with emission regulations.

Detailed Explanation:

1. **Reading Emissions Data**: Hourly emissions data is read for each New England state.
2. **Combining Emissions Data**: Data from individual states is combined into a single dataset.
3. **Cross-Referencing with Facility Data**: Emissions data is filtered to include only active facilities, ensuring consistency with the facility dataset.

Key Code Excerpts:

Listing 9: Combining Emissions Data

```r
combined_emissions_data <- rbindlist(all_emissions_data, fill = TRUE)

# Filter emissions data to match active facilities
active_facilities <- facilities_data_NE$Facility_Unit.ID
combined_emissions_data <- combined_emissions_data[Facility_Unit.ID %in% active_facilities]
```

4.4 New Fossil Fuel Facilities

4.4.1 Overview

The `Fossil_Fuels_New.R` script models the addition of new fossil fuel facilities to meet future energy demands. These facilities are phased in starting from 2033, with three units per facility and one facility added annually.

Detailed Explanation:

1. **Defining Facility Specifications**: Each facility is given a unique ID, and the year it comes online is specified.
2. **Expanding Facility Data**: Each facility is expanded into three units to represent its operational characteristics accurately.

Key Code Excerpts:

Listing 10: Defining New Fossil Fuel Facilities

```
# Define 18 facilities starting in 2033
new_facilities <- data.table(
  Facility_ID = paste0("NGCC", 3000:(3000 + num_facilities - 1)),
  Year_Online = start_year:(start_year + num_facilities - 1)
)

# Expand each facility into 3 units
new_facility_units <- new_facilities[, .(
  Unit_ID = 1:3,
  Facility_ID,
  Year_Online
)]
```

4.5 Imports

4.5.1 Overview

The `imports.R` script processes electricity imports data to calculate statistics and capacity factors for modeling electricity imports into the region. Key components of this script include:

- Loading daily imports data for the years 2011 to 2023 from an Excel file.
- Extracting the maximum capacity factors for transmission lines based on NREL ATB standards (set to 95%).
- Computing percentiles for electricity imports over the given time period.

4.5.2 Detailed Explanation

1. **Loading Libraries**: The script uses a range of libraries, including `dplyr`, `readxl`, `lubridate`, `data.table`, and `tidyverse` for data manipulation, reading Excel files, and date handling.
2. **Defining Time Range**: The imports data is analyzed for the period between January 1, 2011, and December 31, 2023.
3. **Loading Data**:
   - The script identifies and reads sheets from the Excel file corresponding to years between 2011 and 2023.
   - Data from each sheet is combined into a single dataset, appending a `Year` column for clarity.
4. **Percentile Calculations**: The `calculate_percentiles` function calculates the percentiles (from 1% to 99%) of a specified column in the dataset. This is useful for understanding the distribution of electricity imports.

4.5.3 Key Code

Defining Time Range and Max Capacity Factor The time range for the analysis and the maximum capacity factor for transmission lines (set to 95%) are defined as follows:

Listing 11: Defining Time Range and Max Capacity Factor

```
start_date <- as.Date("2011-01-01")
end_date <- as.Date("2023-12-31")
max_CF_transmission_lines <- 0.95 # According to NREL ATB
```

Loading Imports Data The imports data is loaded from an Excel file, extracting only the sheets corresponding to the years of interest (2011–2023):

Listing 12: Loading Electricity Imports Data

```r
file <- "daily_capacity_status.xlsx"
imports_data <- data.frame()
years <- as.character(2011:2023)
sheet_names <- excel_sheets(file)
sheet_names_of_interest <- sheet_names[sapply(sheet_names, function(name) name %in% years)]

for (sheet_name in sheet_names_of_interest) {
  sheet_data <- read_excel(file, sheet = sheet_name)
  sheet_data$Year <- sheet_name
  imports_data <- rbind(imports_data, sheet_data)
}
```

Calculating Percentiles The following function calculates percentiles for a specified column of the imports dataset:

Listing 13: Percentile Calculation Function

```r
calculate_percentiles <- function(data, column) {
  quantiles <- quantile(data[[column]], probs = seq(0.01, 0.99, by = 0.01), na.rm = TRUE)
  tibble(Percentile = seq(1, 99), Value = quantiles)
}
```

This function is applied to the imports data to understand the variability and trends in electricity imports.

4.5.4 Insights and Results

The `imports.R` script prepares the data for modeling electricity imports, providing:

- A unified dataset of daily electricity imports from 2011 to 2023.
- Insights into the statistical distribution of imports through percentile calculations.
- Capacity factor assumptions for transmission infrastructure.

4.6 Demand Analysis

4.6.1 Overview

The script `demand.R` processes hourly demand data to ensure consistency and accuracy, especially for leap years. Its main functionalities include:

- Estimating missing February 29th demand values in leap years.
- Cleaning and standardizing hourly demand data for the energy model.

4.6.2 Leap Year Adjustment

The script checks for leap years and fills missing February 29th demand data by averaging February 28th and March 1st values for each hour. The following code performs this operation:

Listing 14: Filling Missing Leap Year Data

```r
fill_leap_year_data <- function(data) {
  for (year in unique(data$Year)) {
    if (year %% 4 == 0 && (year %% 100 != 0 || year %% 400 == 0)) {  # Check if leap year
      for (hour in 0:23) {
        feb_28_data <- data[Year == year & Month == 2 & Day == 28 & Hour == hour, Demand]
        mar_1_data <- data[Year == year & Month == 3 & Day == 1 & Hour == hour, Demand]

        if (length(feb_28_data) == 1 && length(mar_1_data) == 1) {
          leap_day_demand <- mean(c(feb_28_data, mar_1_data), na.rm = TRUE)
          data <- rbind(data, data.table(
            Year = year,
            Month = 2,
            Day = 29,
            Hour = hour,
            Demand = leap_day_demand
          ))
        }
      }
    }
  }
  return(data)
}
```

4.7 Randomization

4.7.1 Overview

The `randomization.R` script generates pseudo-random sequences for use in probabilistic modeling. These sequences are designed to represent variability in key parameters or inputs to the model. The script performs the following tasks:

- Defines parameters for the randomization process, including the number of sequences and the length of each sequence.
- Uses a custom function to generate pseudo-random sequences.
- Saves the generated sequences to a CSV file for integration into the modeling framework.

4.7.2 Detailed Explanation

1. **Loading Libraries**: The `randtoolbox` library is loaded, which provides tools for generating random and quasi-random numbers.
2. **Defining Parameters**:
   - `num_sequences`: The total number of sequences to generate (1,000,000).
   - `num_numbers`: The length of each sequence (250 numbers per sequence).

   These parameters define the scope and granularity of the randomization process.
3. **Generating Pseudo-Random Sequences**: A custom function, `generate_pseudo_random_sequences`, is defined to generate sequences of pseudo-random numbers. This function uses the `sample` function to create a sequence of integers between 1 and 99.
4. **Saving Random Sequences**: The generated sequences are saved to a CSV file for later use in the generation expansion model.

4.7.3 Key Code

Defining the Randomization Function The following function generates pseudo-random sequences:

Listing 15: Pseudo-Random Sequence Generator

```
generate_pseudo_random_sequences <- function(num_sequences, num_numbers) {
  random_sequences <- replicate(num_sequences, sample(1:99, num_numbers, replace = TRUE), ...
      simplify = FALSE)
  return(random_sequences)
}
```

Generating and Transposing Random Sequences The generated sequences are combined into a matrix and transposed for storage:

Listing 16: Generating and Transposing Random Sequences

```r
# Parameters
num_sequences <- 1e6
num_numbers <- 250

# Generate Pseudo-random sequences
Percentile_sequences_random <- generate_pseudo_random_sequences(num_sequences, num_numbers)

# Convert the list to a matrix and transpose it
Percentile_sequences_matrix <- t(do.call(rbind, Percentile_sequences_random))
```

Saving to a CSV File The pseudo-random sequences are saved to a CSV file for subsequent use:

Listing 17: Saving Random Sequences to CSV

```r
file_path_random_csv <- "Random_Sequence.csv"
write.csv(Percentile_sequences_matrix, file_path_random_csv, row.names = FALSE)
```

4.7.4 Insights and Applications

The randomization process enables:

- Creation of large-scale probabilistic datasets for use in energy modeling.
- Representation of uncertainty and variability in model parameters.
- Efficient storage of sequences for repeated use across simulations.

The generated pseudo-random sequences are integrated into the generation expansion model, adding flexibility and robustness to the analysis.

5 Dispatch Curve Results Processing

Once the core dispatch-curve routines have produced hourly and per-unit outputs, we run end-to-end simulations across all decarbonization pathways, aggregate the results, export them, and generate basic diagnostic plots.

5.1 5.1 Simulation Execution and Aggregation

We iterate over each simulation index and each pathway, calling in turn:

1. `dispatch_curve(sim, pathway)` to compute the clean-plus-storage dispatch curve,
2. `dispatch_curve_adjustments()` to sample and adjust fossil-fuel outputs under ramp constraints,
3. `dispatch_curve_calibrations()` to recombine fossil outputs and re-run battery/import adjustments.

Each function returns a data.table tagged with `Simulation` and `Pathway`. We then flatten the nested lists into two master tables: one for the final hourly dispatch results and one for the facility-level fossil-fuel outputs.

Listing 18: Running simulations and combining results

```r
# 1. Define simulations & pathways
pathways      <- unique(Hourly_Installed_Capacity$Pathway)
n_simulations <- 1

# 2. Execute each sim and pathway
simulation_results <- lapply(1:n_simulations, function(sim) {
  lapply(pathways, function(pathway) {
    dc_results   <- dispatch_curve(sim, pathway)
    ff_adjusted  <- dispatch_curve_adjustments(dc_results)
    final_hourly <- dispatch_curve_calibrations(dc_results, ff_adjusted)
    # Tag with sim & pathway
    dc_results$Simulation <- ff_adjusted$Simulation <- final_hourly$Simulation <- sim
    dc_results$Pathway    <- ff_adjusted$Pathway    <- final_hourly$Pathway    <- pathway
    list(final_hourly = final_hourly, fossil_fuels = ff_adjusted)
  })
})

# 3. Flatten into two data.tables
combined_final_hourly_results <- do.call(rbind, lapply(simulation_results, function(s) {
  do.call(rbind, lapply(s, `[[`, "final_hourly"))
}))
combined_fossil_fuels_results <- do.call(rbind, lapply(simulation_results, function(s) {
  do.call(rbind, lapply(s, `[[`, "fossil_fuels"))
}))
```

5.2 5.2 Exporting Results

After aggregation, we write the two tables to CSV for downstream analysis or visualization.

Listing 19: Writing aggregated results to CSV

```
# Save combined hourly dispatch results
write.csv(combined_final_hourly_results,
          "/path/to/Hourly_Results_NE.csv",
          row.names = FALSE)

# Save combined facility level fossil outputs
write.csv(combined_fossil_fuels_results,
          "/path/to/Facility_Level_Results.csv",
          row.names = FALSE)
```

6 Dispatch Curve Results Processing

6.1 Overview

After generating hourly dispatch and fossil-fuel outputs, we perform two post-processing steps:

1. **Spatial enrichment:** assign each facility to its county (and state) via a spatial join against U.S. Census boundaries, fixing Connecticut planning regions manually where needed.
2. **Annual system summaries:** compute key metrics—renewable penetration, battery discharge, shortages, and CO_2 emissions—by year, simulation, and pathway, then combine across all scenarios.

6.2 Spatial Enrichment of Facility Results

1. Read the facility-level summary CSV into a `data.table`, drop any phantom columns.
2. Filter out rows missing latitude/longitude, convert to an `sf` object (WGS84).
3. Load county geometries (via `tigris::counties` plus a New England GeoJSON), ensure both use CRS 4326.
4. Spatially join facilities to counties (`st_within`), then:
   - For CT, map planning-region names to county names and GEOIDs.
   - Manually correct any unmatched units (e.g. facility IDs “10823_S42”, “10823_S43” to Suffolk, MA).
5. Save the enriched table (with `County_State`, `GEOID`, etc.) to CSV.

6.3 Annual System Summaries

1. For each hourly-results CSV, compute:

$$\text{RenewablePenetration} = \frac{\text{Solar\_MWh} + \text{Onshore\_MWh} + \text{Offshore\_MWh}}{\text{Clean\_MWh} + \text{Imports\_MWh} + \text{Fossil\_MWh} + \text{NewFossil\_MWh}},$$

$$\text{BatteryDischarges\_GWh} = \sum(\text{Calibrated\_Battery\_discharge})/10^3,$$

$$\text{Shortages\_TWh} = \sum(\text{Calibrated\_Shortage\_MWh})/10^6,$$

$$\text{CO2\_tons} = \sum(\text{CO2\_tons}).$$

2. Summarize these by **`Year`**, **`Simulation`**, and **`Pathway`**.

3. Row-bind across all files and write `Battery_Data.csv`.

6.4 Key Code Snippets

Listing 20: Spatial join and county assignment

```r
#--- Load and clean facility results
Yearly_Fac <- fread(file_path)[, V1 := NULL]
fac_clean  <- Yearly_Fac[!is.na(latitude) & !is.na(longitude)]
fac_sf     <- st_as_sf(fac_clean, coords=c("longitude","latitude"), crs=4326)

#--- Load counties, spatial join
counties   <- counties(cb=TRUE, class="sf") %>% st_transform(4326)
fac_sf     <- st_join(fac_sf, counties, join=st_within)

#--- Fix CT planning regions, county names and GEOIDs
fac_dt <- as.data.table(fac_sf)
fac_dt[STUSPS=="CT", NAME := planning_to_county[NAME]]
fac_dt[STUSPS=="CT", GEOID := county_to_geoid[NAME]]
#--- Manual fixes for known mismatches...
fwrite(fac_dt, file.path(output_path,"Yearly_Facility_Level_Results_County_added_in.csv"))
```

Listing 21: Annual summary of renewables, storage, shortages, CO2

```r
#--- Read all hourly outputs, compute per-file summaries
files        <- list.files(folder_path, "^Hourly_Results_.*\\.csv$", full.names=TRUE)
summary_list<- list()
for(f in files) {
  dt <- fread(f)
  dt[, Renewable.Penetration :=
       (Solar_MWh+Onshore_MWh+Offshore_MWh)/
       (Calibrated_Total_import_net_MWh+Old_Fossil_Fuels_adj_MWh+Clean_MWh+New_Fossil_Fuel_MWh)]
  summary_list[[f]] <- dt[, .(
      RP_mean                = mean(Renewable.Penetration,na.rm=TRUE),
      RP_max                 = max(Renewable.Penetration,na.rm=TRUE),
      RP_min                 = min(Renewable.Penetration,na.rm=TRUE),
      Battery_Discharges_GWh = sum(Calibrated_Battery_discharge,na.rm=TRUE)/1e3,
      Shortages_TWh          = sum(Calibrated_Shortage_MWh,na.rm=TRUE)/1e6,
      CO2_tons               = sum(CO2_tons,na.rm=TRUE)
    ), by=.(Year,Simulation,Pathway)]
}
combined_summary <- rbindlist(summary_list)
write.csv(combined_summary,
          file.path(output_path,"Battery_Data.csv"),
          row.names=FALSE)
```

7 Total Costs

7.1 CAPEX, FOM, and VOM Costs

7.1.1 CAPEX and FOM for Fossil Energy

The `1_CAPEX_FOM_ATB_Fossil.R` script calculates capital expenditures (CAPEX) and fixed operation and maintenance costs (FOM) for fossil fuel-based facilities. It performs the following tasks:

- Loads ATB (Annual Technology Baseline) cost data for fossil fuels.
- Calculates net present value (NPV) for CAPEX and FOM using a discount rate of 2%.
- Extracts data for various fossil fuel types, such as coal, gas, oil, and wood.

Key Code:

Listing 22: Calculating NPV for Fossil CAPEX and FOM

```r
calculate_npv <- function(dt, rate, base_year) {
  npv <- sum(dt[,2] / (1 + rate)^(dt[,1] - base_year))
  return(npv)
}

discount_rate <- 0.02
base_year <- 2023

Coal_NPC <- Fossil_Fuels_NPC[grepl("Coal", Primary.Fuel.Type, ignore.case = TRUE)]
Gas_CC_NPC <- Fossil_Fuels_NPC[grepl("Gas", Primary.Fuel.Type, ignore.case = TRUE) &
                                grepl("Combined", Unit.Type, ignore.case = TRUE)]
```

7.1.2 CAPEX and FOM for Non-Fossil Energy

The `2_CAPEX_FOM_ATB_Non_Fossil.R` script focuses on renewable and nuclear energy sources. It:

- Loads ATB data for non-fossil technologies, including solar, onshore wind, offshore wind, nuclear, and hydro.
- Calculates CAPEX and FOM for each technology across different scenarios (Advanced, Moderate, Conservative).

Key Code:

Listing 23: Loading and Setting Non-Fossil Data

```r
ATBe <- fread("ATBe.csv")
path <- "Installed Capacity and CF Non Fossil"
files <- list.files(path = path, pattern = "\\.csv$", full.names = TRUE)

datasets <- list()
for (file in files) {
  dataset_name <- gsub(".*/|\\.csv$", "", file)
  datasets[[dataset_name]] <- fread(file)
}

list2env(datasets, envir = .GlobalEnv)  # Release datasets into the environment
setDT(Solar_NPC)
setDT(Nuclear_hydro_bio_NPC)
```

7.1.3 CAPEX and FOM for Imports

The `3_CAPEX_FOM_Imports.R` script calculates CAPEX and FOM costs for import energy capacity. Key operations include:

- Loading capacity data for imports across three scenarios (S1, S2, S3).
- Calculating new capacity built each year and the associated costs.

Key Code:

Listing 24: Loading and Calculating Import Capacity

```
Imports_S1 <- read_excel("Import Capacity.xlsx", sheet = "S1")
Imports_S2 <- read_excel("Import Capacity.xlsx", sheet = "S2")
Imports_S3 <- read_excel("Import Capacity.xlsx", sheet = "S3")

Imports_Capacity <- rbindlist(list(Imports_S1, Imports_S2, Imports_S3), use.names = TRUE, ...
    fill = TRUE)
Imports_Capacity[, new_capacity := QC - shift(QC, 1, type = "lag"), by = Scenario]
```

7.1.4 VOM for Fossil Energy

The `4_VOM_ATB_Fossil.R` script calculates variable operation and maintenance (VOM) costs for fossil fuels:

- Merges fossil fuel generation data with facility-specific data.
- Calculates yearly VOM costs for different fossil fuel technologies, accounting for scenarios in the ATB data.

Key Code:

Listing 25: Calculating VOM Costs for Fossil Fuels

```
Yearly_Facility_Level_Results <- selected_cols[Yearly_Facility_Level_Results, on = ...
    "Facility_Unit.ID"]
Yearly_Facility_Level_Results[, VOM_Cost := Generation * VOM_Rate, by = Unit.Type]
```

7.1.5 VOM for Non-Fossil Energy

The `5_VOM_ATB_Non_Fossil.R` script calculates VOM costs for renewable and nuclear energy. It:

- Processes VOM costs for solar, wind, hydro, and nuclear technologies.
- Filters data based on ATB scenarios and technology-specific parameters.

Key Code:

Listing 26: Processing VOM Costs for Non-Fossil Technologies

```
process_non_fossil <- function(technology, techdetail, dataset, column_name, simulation, ...
    scenario) {
  dataset <- dataset[Technology == technology & TechDetail == techdetail]
  dataset[, VOM_Cost := Generation * VOM_Rate]
  return(dataset)
}
```

—

7.1.6 Insights and Applications

These scripts collectively provide comprehensive cost assessments for CAPEX, FOM, and VOM across fossil, non-fossil, and imported energy sources. The outputs are critical for evaluating the economic feasibility of different decarbonization pathways under various scenarios.

7.2 Fuel Costs

7.2.1 Fuel Costs for Fossil Energy

The `7_Fuel_ATB_Fossil.R` script calculates fuel costs for fossil fuel energy sources using the NREL ATB (Annual Technology Baseline) dataset. Key operations include:

- Loading CPI (Consumer Price Index) data via the `fredr` API to adjust costs to 2024 dollars.
- Loading ATB fuel cost data for fossil fuels, including coal, natural gas, and oil.
- Calculating Net Present Value (NPV) of fuel costs using a 2% discount rate.

Key Code:

Listing 27: Loading and Adjusting Fuel Costs for Fossil Fuels

```r
# Set discount rate and base year
discount_rate <- 0.02
base_year <- 2023

# Load CPI data to calculate conversion rate
cpi_data <- fredr(series_id = "CPIAUCSL", observation_start = as.Date("2000-01-01"), ...
    observation_end = as.Date("2024-01-01"))
cpi_2021 <- filter(cpi_data, year(date) == 2021) %>% summarise(YearlyAvg = mean(value))
cpi_2024 <- filter(cpi_data, year(date) == 2024) %>% summarise(YearlyAvg = mean(value))
conversion_rate <- cpi_2024$YearlyAvg / cpi_2021$YearlyAvg

# Load ATB fuel cost data
file_path <- "ATB_2021_Fuel_Costs.csv"
ATB_Fuel_Costs <- fread(file_path)
ATB_Fuel_Costs[, Adjusted_Cost := Original_Cost * conversion_rate]
```

This script adjusts fossil fuel costs to 2024 dollars and calculates NPVs to evaluate long-term fuel expenditure.

7.2.2 Fuel Costs for Non-Fossil Energy

The `8_Fuel_ATB_Non_Fossil.R` script processes fuel costs for non-fossil energy sources such as nuclear, hydro, and biomass. Its primary functions include:

- Loading ATB cost data for non-fossil fuel types.
- Filtering data based on technology, scenarios, and simulation parameters.
- Calculating VOM (Variable Operation and Maintenance) costs where applicable.

Key Code:

Listing 28: Processing Non-Fossil Fuel Costs

```r
process_non_fossil <- function(technology, techdetail, dataset, column_name, simulation, ...
    scenario) {
  dataset <- dataset[Technology == technology & TechDetail == techdetail]
  dataset[, Adjusted_Cost := Fuel_Cost * VOM_Rate]
  return(dataset)
}

# Example: Processing nuclear fuel costs
process_non_fossil("Nuclear", "Advanced", ATBe, "Fuel_Cost", "Simulation1", "Scenario1")
```

This script enables scenario-based analysis of non-fossil fuel costs under varying economic and technological assumptions.

7.2.3 Insights and Applications

These scripts provide a detailed analysis of fuel costs for fossil and non-fossil energy sources:

- Fossil fuel costs are adjusted to 2024 dollars using CPI data and evaluated for long-term feasibility.
- Non-fossil fuel costs are integrated into scenario-based models to assess their economic competitiveness.

The outputs support informed decision-making in decarbonization pathway modeling.

7.3 Imports Costs

7.3.1 Overview

The `6_Imports.R` script calculates costs related to electricity imports. Key functionalities include:

- Adjusting costs to 2024 dollars using Consumer Price Index (CPI) data retrieved from the `fredr` API.
- Calculating Net Present Value (NPV) of imports costs using a discount rate of 2%.
- Processing capacity data across multiple scenarios to estimate annual costs.

7.3.2 Detailed Explanation

1. **Loading CPI Data**:
   - The script retrieves CPI data for 2021, 2022, and 2024 to adjust historical costs to 2024 values.
   - Conversion rates are calculated based on the ratio of CPI values.
2. **Loading Capacity Data**:
   - Import capacity data is loaded from an Excel file, with separate sheets representing different scenarios (e.g., S1, S2, S3).
   - New capacity additions are calculated for each year within each scenario.
3. **NPV Calculation**: The script calculates the NPV of costs for each scenario, providing a long-term cost assessment.

7.3.3 Key Code

CPI Adjustments:

Listing 29: Loading and Adjusting CPI Data

```r
# Set API key and retrieve CPI data
fredr_set_key("63522eae4ec927d6f1d9d86bf7826cc8")
cpi_data <- fredr(series_id = "CPIAUCSL", observation_start = as.Date("2000-01-01"), ...
    observation_end = as.Date("2024-01-01"))

# Calculate conversion rates
cpi_2021 <- filter(cpi_data, year(date) == 2021) %>% summarise(YearlyAvg = mean(value))
cpi_2024 <- filter(cpi_data, year(date) == 2024) %>% summarise(YearlyAvg = mean(value))
conversion_rate_2021 <- cpi_2024$YearlyAvg / cpi_2021$YearlyAvg
```

Loading Capacity Data:

Listing 30: Loading Import Capacity Data

```r
path <- "Imports_Capacity.xlsx"

# Load data for different scenarios
Imports_S1 <- read_excel(path, sheet = "S1")
Imports_S2 <- read_excel(path, sheet = "S2")
Imports_S3 <- read_excel(path, sheet = "S3")

```

```r
# Combine and calculate new capacity
Imports_Capacity <- rbindlist(list(Imports_S1, Imports_S2, Imports_S3), use.names = TRUE, ...
    fill = TRUE)
Imports_Capacity[, new_capacity := QC - shift(QC, 1, type = "lag"), by = Scenario]
```

NPV Calculation:

Listing 31: Calculating NPV for Import Costs

```r
calculate_npv <- function(dt, rate, base_year, col) {
  npv <- sum(dt[[col]] / (1 + rate)^(dt[['Year']] - base_year))
  return(npv)
}

discount_rate <- 0.02
base_year <- 2023

# Example NPV calculation
npv_scenario1 <- calculate_npv(Imports_Capacity[Scenario == "S1"], discount_rate, ...
    base_year, "new_capacity")
```

7.3.4 Insights and Applications

This script provides a detailed assessment of import costs:

- Adjusting for inflation ensures that all costs are consistent with 2024 dollars.
- Scenario-based analysis enables a comparison of costs across different import strategies.
- NPV calculations provide a long-term perspective on import-related expenditures.

7.4 GHG Emissions Costs

7.4.1 Overview

The `9_GHG_Emissions.R` script calculates costs associated with GHG emissions. Key functionalities include:

- Loading annual GHG emissions data in metric tons CO_2-equivalent (IPCC AR4 GWP).
- Interpolating social costs for CO_2, CH_4, and N_2O from 2025 to 2050.
- Adjusting those costs to 2024 USD via CPI data from the `fredr` API.
- Calculating the Net Present Value (NPV) of total GHG costs at a 2.5 % discount rate.

7.4.2 Detailed Explanation

1. **Loading Emissions Data** Read facility-level and system-level CSVs of annual GHG emissions (all in metric tons CO_2-eq).
2. **CPI Adjustment** Fetch CPI series (2000–2024) via `fredr`, compute the 2020→2024 conversion factor, and scale all future SCC values.
3. **Cost Interpolation** Define SCC endpoints in 2025 and 2050 for CO_2, CH_4, N_2O; linearly interpolate for each year 2025–2050.
4. **NPV Calculation** For each simulation and pathway, sum annual

$$\text{Cost}_t = \text{CO}_{2,t} \times c_{\text{CO}_2,t} + \text{CH}_{4,t} \times c_{\text{CH}_4,t} + \text{N}_2\text{O}_t \times c_{\text{N}_2\text{O},t}$$

then discount from 2025–2050 at 2.5 %: $\text{NPV} = \sum_{t=2025}^{2050} \frac{\text{Cost}_t}{(1+0.025)^{t-2024}}$.

7.4.3 Key Code

CPI cost curve:

```
discount_rate <- 0.025
base_year     <- 2024

fredr_set_key("63522eae4ec927d6f1d9d86bf7826cc8")
cpi <- fredr("CPIAUCSL", as.Date("2000-01-01"), as.Date("2024-01-01"))
f20 <- mean(filter(cpi, year(date)==2020)$value)
f24 <- mean(filter(cpi, year(date)==2024)$value)
rate <- f24/f20

cost25 <- list(CO2=130, CH4=1590, N2O=39972)*rate
cost50 <- list(CO2=205, CH4=3547, N2O=65635)*rate

interp_cost <- function(y, start=2025, end=2050, c0, c1) {
  c0 + (c1-c0)*(y-start)/(end-start)
}
GHG_Costs <- data.table(Year=2025:2050)[
  , `:=`(
    CO2_cost = interp_cost(Year, c0=cost25$CO2, c1=cost50$CO2),
    CH4_cost = interp_cost(Year, c0=cost25$CH4, c1=cost50$CH4),
    N2O_cost = interp_cost(Year, c0=cost25$N2O, c1=cost50$N2O)
  )]
```

NPV calculation:

```
# after merging annual total_CO2, total_CH4_eq, total_N2O_eq and GHG_Costs:
dt[, annual_cost := total_CO2*CO2_cost + total_CH4_eq*CH4_cost + total_N2O_eq*N2O_cost]
calc_npv <- function(d) sum(d$annual_cost/(1+discount_rate)^(d$Year-base_year))

# per simulation & pathway
npv_list <- dt[, .(NPV=calc_npv(.SD)), by=.(Simulation, Pathway)]

fwrite(npv_list[, .(
  NPV_min  = min(NPV),
  NPV_mean = mean(NPV),
  NPV_max  = max(NPV)
), by=Pathway],
  "GHG_Emissions.csv")

fwrite(npv_list, "GHG_Emissions_per_simulation.csv")
```

7.5 Air Pollutant Emissions Costs

7.5.1 Overview

The `10_Air_emissions.R` script calculates costs associated with air pollutant emissions. Key functionalities include

- Converting emission quantities to metric tons for consistency.
- Adjusting costs to 2024 dollars using Consumer Price Index (CPI) data from the `fredr` API.
- Estimation of the net present value (NPV) of the air pollutant costs using a discount rate of 2%.

7.5.2 Detailed Explanation

1. **Emission unit conversions**:
   - Emission data are converted from pounds (lbs) to US tons, and subsequently to metric tons for compatibility with international standards.
2. **CPI Adjustments**: Costs are adjusted to 2024 dollars by calculating a conversion rate between CPI values from 2000 and 2024.

3. **NPV Calculation**: The script calculates the NPV of air pollutant costs, enabling long-term economic analysis.

7.5.3 Key Code

Unit Conversions:

Listing 32: Converting Emission Units to Metric Tons

```r
lbs_tons_conversion <- 1 / 2204.62  # lbs to tons
ton_conversion <- 0.907185  # US tons to metric tons

# Example conversion
emission_data[, Metric_Tons := Emissions_Lbs * lbs_tons_conversion * ton_conversion]
```

CPI Adjustments:

Listing 33: Adjusting Costs Using CPI Data

```r
fredr_set_key("63522eae4ec927d6f1d9d86bf7826cc8")
cpi_data <- fredr(series_id = "CPIAUCSL", observation_start = as.Date("2000-01-01"), ...
    observation_end = as.Date("2024-01-01"))

# Calculate conversion rate
cpi_2000 <- filter(cpi_data, year(date) == 2000) %>% summarise(YearlyAvg = mean(value))
cpi_2024 <- filter(cpi_data, year(date) == 2024) %>% summarise(YearlyAvg = mean(value))
conversion_rate <- cpi_2024$YearlyAvg / cpi_2000$YearlyAvg

# Adjust costs
emission_data[, Adjusted_Cost := Original_Cost * conversion_rate]
```

NPV Calculation:

Listing 34: Calculating NPV for Air Pollutant Costs

```r
calculate_npv <- function(dt, rate, base_year, col) {
  npv <- sum(dt[[col]] / (1 + rate)^(dt[["Year"]] - base_year))
  return(npv)
}

discount_rate <- 0.02
base_year <- 2024

# Example NPV calculation
npv_air_emissions <- calculate_npv(emission_data, discount_rate, base_year, "Adjusted_Cost")
```

7.5.4 Insights and Applications

This script supports air pollutant cost modeling by:

- Ensuring consistency in emission quantities via unit conversions.
- Adjusting historical costs to 2024 values for accurate comparisons.
- Providing a long-term economic perspective through NPV calculations.

These analyses are essential for assessing the economic impact of air pollutant emissions and supporting policy-making in decarbonization strategies.

7.6 Unmet Demand Penalty Costs

7.6.1 Overview

The `11_Unmet_Demand.R` script now:

- Loads annual shortage results from a CSV (`Yearly_Results_Shortages.csv`).

- Converts unmet-demand (MWh) into dollar penalties using an EVOLL of \$3500/MWh (2024 USD).
- Computes NPVs of those penalties, discounting at 2.5 % from a 2024 base year, for each simulation and pathway.
- Exports per-simulation NPVs and pathway-level summaries to CSV.

7.6.2 Detailed Explanation

1. **Load shortage data:** Read `Yearly_Results_Shortages.csv` into a `data.table`.
2. **Compute dollar penalties:**

   $$\text{Unmet\_Demand\_USD\_total} = \text{Unmet\_Demand\_total\_MWh} \times 3500$$

   (3500 \$/MWh in 2024 USD).
3. **NPV calculation:** Define

   $$\text{NPV} \;=\; \sum_t \frac{\text{Unmet\_Demand\_USD\_total}_t}{(1+0.025)^{\,t-2024}}$$

   and compute it for each (`Simulation`, `Pathway`).
4. **Export:** Write two CSVs:
   - `Unmet_demand.csv`: per-simulation NPVs (excludes zeros).
   - (Optionally) summary statistics by pathway.

7.6.3 Key Code

Listing 35: Unmet-Demand Penalty NPV

```r
library(data.table)
# NPV helper
calculate_npv <- function(dt, rate, base_year, col) {
  sum(dt[[col]]/(1+rate)^(dt$Year-base_year))
}

discount_rate <- 0.025
base_year     <- 2024

# EVOLL in $/MWh (2024 USD)
EVOLL_2025 <- 3500

# Load shortage results
file_path     <- "Yearly_Results_Shortages.csv"
output_path   <- "   /Total_Costs_Results"
Yearly_Results <- fread(file_path)

# Compute dollar penalties
Yearly_Results[, Unmet_Demand_USD_total := Unmet_Demand_total_MWh * EVOLL_2025]

#  P e r sim  and pathway NPV
npv_list <- list()
for(sim in unique(Yearly_Results$Simulation)) {
  for(path in unique(Yearly_Results$Pathway)) {
    dt  <- Yearly_Results[Simulation==sim & Pathway==path]
    npv <- calculate_npv(dt, discount_rate, base_year, "Unmet_Demand_USD_total")
    if(npv!=0) {
      npv_list[[paste(sim,path)]] <- data.table(Simulation=sim, Pathway=path, NPV=npv)
    }
  }
}
```

```
combined_npvs <- rbindlist(npv_list)

# Summary by pathway (in billions)
combined_npvs[, .(
  mean_NPV = mean(NPV)/1e9,
  sd_NPV   = sd(NPV)/1e9
), by=Pathway]

# Save results
fwrite(combined_npvs, file.path(output_path,"Unmet_demand.csv"))
```

7.7 Beyond the Fence Line Costs

7.7.1 Overview

The scripts `12_Other_S3_CAN_Hydro_CostAssumptions.R` and `12_Other_S3_CAN_Hydro.R` now:

- Read decarbonization pathways to compute Quebec import capacity differences (B3vsB1).
- Derive annual hydropower build (ΔMW) from the cumulative import-capacity difference.
- Fetch CPI (2016–2024) to convert ISO-NE CAPEX ($5537/kW) and ATB cost ranges into 2024 USD.
- Compute annual CAPEX, fixed OM (FOM), and variable OM (VOM) costs.
- Estimate CH emissions from imports (Delwiche et al. EF), project CH social costs (2025–2050), and compute NPVs.

7.7.2 Hydropower Capacity Build

- QC capacity difference: $\Delta_{\text{QC}}(t) = \text{cumsum}[\,\text{QC}_{\text{B3}}(t) - \text{QC}_{\text{B1}}(t)\,]$.
- Base hydro capacity: $H(t) = \Delta_{\text{QC}}(t)/\text{CF}$.
- Year-on-year build: $\text{Build}(t) = \max\{0, H(t) - H(t-1)\}$.

7.7.3 Cost Calculations

- CPI rate: $r = \text{CPI}_{2024}/\text{CPI}_{2016}$.
- Unit CAPEX: $\text{CAPEX}_{\text{unit}} = \$5537/\text{kW} \times 1000 \times r$.
- FOM rate: $\{1.5\%, 2.5\%\} \times \text{CAPEX}_{\text{unit}}$.
- VOM: $0.58/MWh.
- Annual costs:
$$\begin{aligned} C_{\text{CAPEX}}(t) &= \text{Build}(t) \times \text{CAPEX}_{\text{unit}}, \\ C_{\text{FOM}}(t) &= \text{QC\_cap\_MW}(t) \times \text{FOM\_rate}/\text{CF}, \\ C_{\text{VOM}}(t) &= \text{QC\_MWh}(t) \times \text{VOM}/\text{CF}. \end{aligned}$$

7.7.4 CH_4 Emissions and NPV

- CH (tonnes): $\text{Imports\_QC\_TWh} \times \text{EF} \times \frac{16}{12} \times 10^{-3}$.
- Project CH cost curve (2025–2050) via CPI-adjusted endpoints and linear interpolation.
- NPV at 2.5 %:
$$\text{NPV} = \sum_{t=2025}^{2050} \frac{C_{\text{t}}}{(1+0.025)^{\,t-2024}}.$$
- Export CAPEX/FOM/VOM and CH NPVs by pathway to CSV.

7.8 Total Costs

7.8.1 Overview

The `All_Costs_Tabulated.R` script automates assembling every cost element into two final tables:

- `All_Costs_per_Simulation.csv`: per-simulation, per-pathway NPVs for CAPEX, FOM, VOM, fuel, imports, GHG, air emissions, unmet demand and Canada-side costs, plus total and B1-difference.
- `All_Costs.csv`: across-simulation means by pathway and cost type (CAPEX, Fixed O&M, etc.), after dropping outlier simulations.

7.8.2 Detailed Explanation

1. **Load component CSVs** List all "*.csv" in the results folder, read each with `fread()`, assign by basename.
2. **Summarize each cost category**
   - CAPEX&FOM: bind fossil, non-fossil, imports, group by Simulation+Pathway+scenario, sum NPVs.
   - VOM, Fuel, Imports, GHG, Air, Unmet: similar group-summaries.
   - Canada side: merge hydropower CAPEX/FOM/VOM and CH NPVs with adjusted QC imports.
3. **Merge all** Full-join per-category tables on (Simulation, Pathway).
4. **Split B3** Replace "B3" with "B3(1)" (zero Canada costs) and "B3(2)" (zero imports).
5. **Compute totals** Sum all "_mean_bUSD"/"_max_bUSD"/"_min_bUSD" columns to get per-pathway NPV and B1-differences.
6. **Filter outliers aggregate** Pivot long, compute per-sim total cost, drop sims outside 1.5 × IQR, then average by pathway+cost type.
7. **Save CSVs** Write both detailed and aggregated tables to disk.

7.8.3 Key Code

Listing 36: Core consolidation steps

```r
# 1. Read all cost CSVs
csvs <- list.files(folder_path,"\\.csv$",full.names=TRUE)
tables <- lapply(csvs, fread)
names(tables) <- tools::file_path_sans_ext(basename(csvs))

# 2. Summarize CAPEX & FOM
CAPEX_FOM <- bind_rows(tables$CAPEX_Fixed_Fossil,
                       tables$CAPEX_Fixed_Non_Fossil,
                       tables$CAPEX_FOM_Imports) %>%
  group_by(Simulation,Pathway) %>%
  summarize(across(c(NPV_CAPEX,NPV_FOM), sum), .groups="drop")

# 3. Summarize other costs similarly...
#    (VOM, Fuel, Imports, GHG, Air, Unmet, CAN)

# 4. Merge everything
All_Costs <- reduce(list(CAPEX_FOM, total_VOM, total_Fuel,
                         total_Imports, total_GHG,
                         total_Air, total_Unmet, total_CAN),
                    full_join, by=c("Simulation","Pathway"))

# 5. Split B3     B3(1)/B3(2), compute totals and B1 differences
All_Costs_final <- All_Costs %>%
  # ...split logic...
  rowwise() %>%
  mutate(
    Total_mean = sum(c_across(ends_with("_mean_bUSD")),na.rm=TRUE),
    B1_diff_mean = Total_mean - Total_mean[Pathway=="B1"]
  ) %>%
  ungroup()

# 6. Drop outlier sims and average by pathway+cost type
long <- All_Costs_final %>%
  pivot_longer(cols=ends_with("_bUSD"),
               names_to=c("Type","Stat"),names_pattern="(.*)_(mean|min|max)_bUSD",
               values_to="Value")
# identify outliers by total, then
combined_ranges <- long %>%
  filter(!Simulation %in% outlier_sims) %>%
  group_by(Pathway,Type,Stat) %>%
  summarize(Mean_Value=mean(Value), .groups="drop")

# 7. Write results
fwrite(All_Costs_final, file.path(output_path,"All_Costs_per_Simulation.csv"))
fwrite(combined_ranges, file.path(output_path,"All_Costs.csv"))
```

7.8.4 Insights and Applications

This pipeline:

- Ensures every cost driver is on the same footing and traceable back to its source CSV.
- Produces both granular (per-simulation) and summary (mean by pathway) cost tables.
- Identifies and removes simulation outliers for robust scenario comparison.
- Outputs ready-to-plot data for stacked-bar and error-bar visualizations in `ggplot2`.

7.9 CPI and Other Cost Variables

7.9.1 Overview

The `CPI_Electricity.R` script retrieves Consumer Price Index (CPI) data from the Bureau of Labor Statistics (BLS) API. It focuses on CPI series related to electricity costs across multiple regions. Key functionalities include:

- Retrieving regional CPI data for electricity using the BLS API.
- Structuring the retrieved data into a tabular format.
- Storing the processed data for integration into cost modeling workflows.

7.9.2 Detailed Explanation

1. **API Integration**: The script uses the `blsAPI` and `rjson` libraries to query the BLS API for CPI series data.
2. **Regions and Series**: The script specifies electricity-related series IDs and their corresponding regions, such as New England, Boston, and Middle Atlantic.
3. **Data Processing**: Retrieved data is organized into a data frame with fields for year, period, and CPI values.

7.9.3 Key Code

API Query and Data Retrieval:

Listing 37: Retrieving CPI Data Using BLS API

```r
library(blsAPI)
library(rjson)

api_key <- "a2c1e702f81947118e83cb8d029d6623"
series_ids <- c("APU011072610", "APUS11A72610", "APU012072610", "APUS12A72610", ...
    "APUS12B72610")
regions <- c("New England", "Boston", "Middle Atlantic", "New York", "Philadelphia")

payload <- list(
  'seriesid' = series_ids,
  'regions' = regions,
  'startyear' = '2022',
  'endyear' = '2024',
  'registrationkey' = api_key
)

response <- blsAPI(payload)
json_data <- fromJSON(response)
```

Data Organization:

Listing 38: Organizing CPI Data into a Data Frame

```r
cpi_data <- do.call(rbind, lapply(json_data$Results$series, function(series) {
  data.frame(
    seriesID = series$seriesID,
    Region = series$regions,
    year = sapply(series$data, function(x) x$year),
    period = sapply(series$data, function(x) x$period),
    value = sapply(series$data, function(x) as.numeric(x$value))
  )
}))
```

7.10 Supplemental Costs Calculation

7.10.1 Overview

The `Supplemental_Costs_calc.R` script calculates supplemental costs using GDP, population, and household data. Key functionalities include:

- Loading and filtering population and GDP growth data for the years 2025–2050.
- Converting GDP and household data to 2023 dollars using CPI adjustments.
- Preparing the processed data for integration into demand and cost models.

7.10.2 Detailed Explanation

1. **Data Loading**: Population and GDP data are loaded from an Excel file and filtered for the target years (2025–2050).
2. **CPI Adjustments**: CPI data is retrieved from the FRED API and used to adjust GDP values from 2012 to 2023 dollars.
3. **Data Preparation**: The processed data is stored for further analysis in supplemental cost modeling.

7.10.3 Key Code

Loading and Filtering Data:

Listing 39: Loading Population and GDP Data

```
path <- "/path/to/population_GDP_growth.xlsx"

household_data <- read_excel(path)
household_data <- household_data %>%
  filter(Year ≥ 2025 & Year ≤ 2050)
colnames(household_data) <- c("Year", "n_households_millions", "GDP_bUSD_2012")
```

CPI Adjustments:

Listing 40: Adjusting GDP to 2023 Dollars

```
library(fredr)

fredr_set_key("63522eae4ec927d6f1d9d86bf7826cc8")
cpi_data <- fredr(series_id = "CPIAUCSL", observation_start = as.Date("2000-01-01"), ...
    observation_end = as.Date("2024-01-01"))

cpi_2012 <- filter(cpi_data, year(date) == 2012) %>% summarise(YearlyAvg = mean(value))
cpi_2023 <- filter(cpi_data, year(date) == 2023) %>% summarise(YearlyAvg = mean(value))

conversion_rate <- cpi_2023$YearlyAvg / cpi_2012$YearlyAvg
household_data$GDP_bUSD_2023 <- household_data$GDP_bUSD_2012 * conversion_rate
```

7.10.4 Insights and Applications

These scripts provide critical data processing capabilities:

- **CPI_Electricity.R**: Supports cost modeling by providing up-to-date CPI data for electricity across key regions.
- **Supplemental_Costs_calc.R**: Prepares supplemental cost data, including GDP and household data, adjusted for inflation to 2023 values.

The outputs are essential for integrating economic and demographic variables into energy modeling frameworks.

8 Ecological Impacts

This section documents four Python scripts that quantify ecological impacts—land-use change, avian mortality, water withdrawals, and population exposure—under each decarbonization pathway. Each script follows a Monte Carlo framework to capture uncertainty and produces both a stacked-bar figure with 90

8.1 Land-use Change

Overview `land_use.py` estimates additional habitat loss (million ha) from new capacity (2025→2050) in Solar, Onshore Wind, Canadian Hydro, SMRs and new gas.
Steps

1. Load all pathway sheets and stack into one DataFrame.
2. Inject "Canadian Hydro" additions for B3 manually (0 MW→3692 MW).
3. Compute capacity per technology and pathway.
4. For $n = 1000$ sims, sample land-use factors (fixed, uniform or gamma) per MW, multiply by MW, convert to million ha.
5. Compute per-tech means and total 5/95
6. Plot stacked-bar + asymmetric error bars; save `land_use.png`.
7. Write `land_use_summary.xlsx`.

Key Code

Listing 41: Monte Carlo sampling & bar plot

```
# compute  capacity
df0 = decarb.query("Year==2025").set_index('Pathway')
df1 = decarb.query("Year==2050").set_index('Pathway')
cap_diff = df1[techs] - df0[techs]

# Monte Carlo
samples = np.zeros((n_sims,len(pathways),len(techs)))
for s in range(n_sims):
    # sample factors
    samp = np.random.gamma(alpha,theta,size=len(cap_diff))
    land_diff = cap_diff.values * samp[:,None]
    samples[s,:,:] = land_diff / 1e6

# means & CI
mean_df = pd.DataFrame(samples.mean(0),index=pathways,columns=techs)
low,high = np.percentile(samples.sum(2),[5,95],axis=0)

# plot
mean_df.plot(kind='bar',stacked=True)
plt.errorbar(x, mean_sum, yerr=[mean_sum-low,high-mean_sum],fmt='none')
plt.savefig('land_use.png')
```

8.2 Avian Mortality

Overview `avian_mortality.py` estimates millions of bird and bat deaths from Onshore Wind and Solar capacity additions.

Steps

1. Stack pathway data and inject B3 Hydro as above.
2. Compute MW for Onshore Wind (“bird” and “bat”) and Solar (“bird”).
3. For $n = 1000$, sample mortality rates from Gamma/lognormal distributions.
4. Multiply rates by MW, sum per pathway, convert to millions.
5. Compute mean and 5/95
6. Tabulate per-category and total 10/90

Key Code

Listing 42: Sampling bird/bat mortality

```
for s in range(n_sims):
    bird_on = np.random.gamma(0.20,2.54)
    bat_on  = np.random.gamma(1.538,4.160)
    s_rate  = np.random.lognormal(np.log(1.214),1.409)
    samples[s,:,0] = bird_on * cap_diff["Onshore Wind"]
    samples[s,:,1] = bat_on  * cap_diff["Onshore Wind"]
    samples[s,:,2] = s_rate  * cap_diff["Solar"]
# plotting as above...
```

8.3 Water Withdrawals

Overview `water_use.py` computes water withdrawals (trillion gal) for new SMRs and gas turbines (2025–2050).

Steps

1. Read yearly dispatch results (2025–2050), sum SMRTWh and GasTWh per sim–pathway.
2. Multiply by water-use factors (gallons/MWh), convert to trillion gal.
3. Compute per-pathway mean and 10/90

4. Summarize in `water_withdrawals_summary.xlsx`.

Key Code

Listing 43: Water withdrawals calculation

```
sim_path = yr.groupby(['Simulation','Pathway'])[['SMR_TWh','New_Fossil_Fuel_TWh']].sum()
water = sim_path.mul(pd.Series({'SMR_TWh':740,'New_Fossil_Fuel_TWh':32}))
water_tr = water/1e6  #     trillion gallons
mean_tech = water_tr.groupby('Pathway').mean()
low = water_tr.groupby('Pathway').quantile(0.10)
high= water_tr.groupby('Pathway').quantile(0.90)
# plot as before...
```

8.4 Population Exposure (Viewshed)

Overview `viewshed_exposure.py` estimates how many people (million) live within project buffer zones for new imports, offshore wind and solar.

Steps

1. Compute MW for each of: Imports QC, Imports NYISO, Offshore Wind, Solar.
2. Load New England and source-region polygons; open LandScan raster.
3. For $n = 10$ sims and each pathway×tech: sprinkle one point per 1 GW, buffer by tech-specific radius, mask raster, sum population.
4. Convert to million people; compute mean and 10/90
5. Plot stacked-bar + CI; save `population_exposure.png`.
6. Save Excel summary `viewshed_exposure_summary.xlsx`.

Key Code

Listing 44: Monte Carlo exposure via raster masking

```
for s in range(n_mc):
  for i,pw in enumerate(pathways):
    for j,tech in enumerate(techs):
      Δ = cap_diff_all.at[pw,tech]
      n_proj = int(np.ceil(Δ/proj_size[tech]))
      # random points in region...
      for p in pts:
        buf = p.buffer(buffers_m[tech])
        arr,_ = rasterio.mask.mask(src,[buf],crop=True,nodata=0)
        pop_sum += arr.sum()
      mc[s,i,j] = pop_sum/1e6
```

8.4.1 Insights

- **Land-use:** Solar and onshore wind drive the largest habitat conversion, with B3 highest overall.
- **Avian mortality:** Onshore wind causes several million bird and bat fatalities; Solar adds less but non-negligible.
- **Water withdrawals:** SMR cooling dominates water use vs. dry-cooled gas.
- **Population exposure:** Imports and offshore wind buffers potentially affect millions of residents in New England.

This is some text.