PyPSA-SPICE Model Builder

Documentation

Data and Modelling Team at Agora Think Tanks

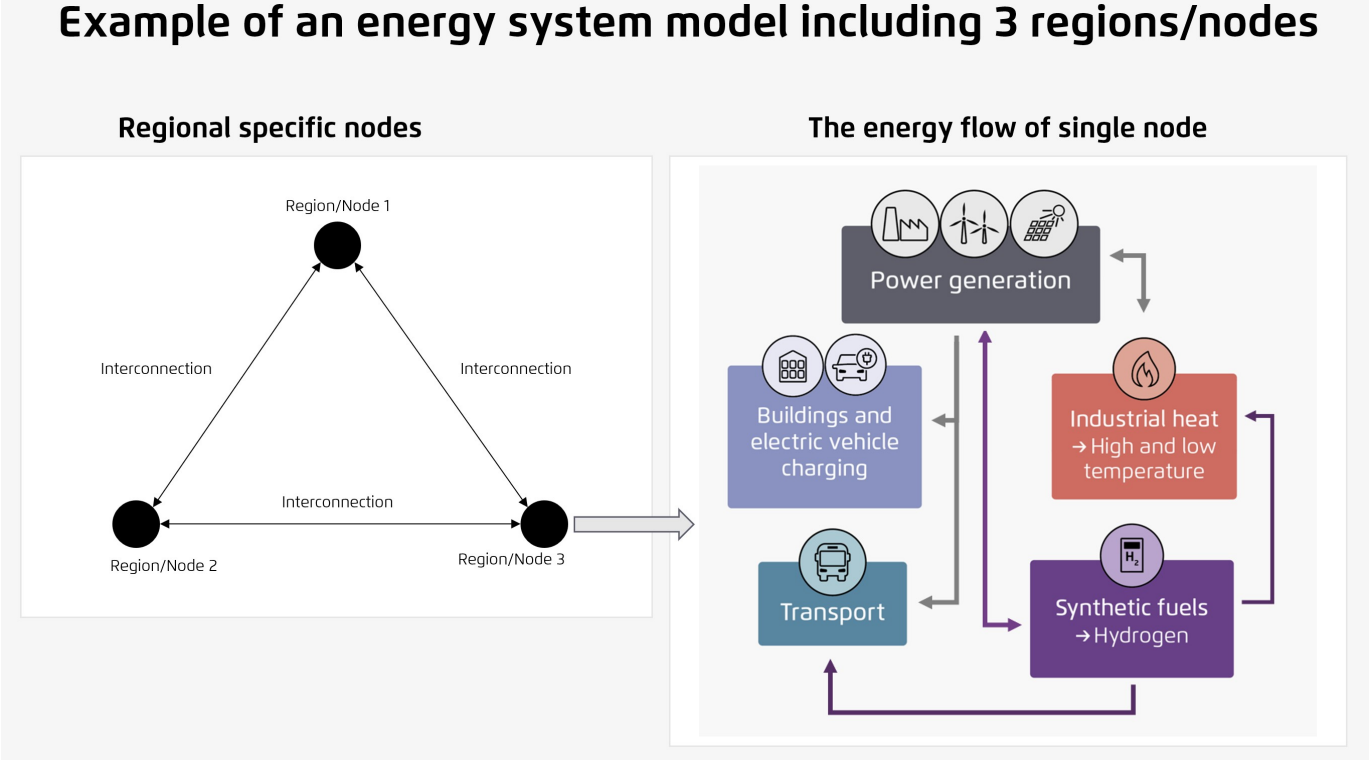
Table of contents

1. PyPSA-SPICE: Python-based Sector-Coupled Optimisation for Pathways Exploration

> **Info**
>
> If you are considering using this model builder, please reach out to us at Email_to_be_added. We would be happy to help you get started. If you encounter a bug, please create a new issue. For new ideas or feature requests, you can start a conversation in the discussions section of the repository.

PyPSA-SPICE is an open-source model builder for assessing national mid/long-term energy scenarios using a least-cost, multi-sectoral optimization approach based on the PyPSA framework. It can be used to build models that represent one or more countries across multiple interconnected nodes linked by electricity transmission, and within each region, it models the integration of power, heat, and transport sectors. The model focuses on the power sector, which is represented with a high level of detail.

The model workflow has been designed to be more accessible compared to other PyPSA-based models, though basic Python coding knowledge is required.



1.1 Key Features of PyPSA-SPICE

- Assessment of national or regional long-term energy scenarios using a least-cost optimization approach based on the PyPSA framework.
- Co-optimization of generation, capacity, and interconnector expansion at hourly resolution.
- Power plants are represented at technology resolution, with user-defined clustering within technologies as needed.
- Straightforward model creation for new countries and/or regions defined by the user.
- Easy integration of custom data into the model.

- Several pre-defined custom constraints including energy independence, reserve margin, and must-run constraints on thermal generators.
- Flexible sectoral coverage: base power sector model can be complemented with industry and transport sectors for full energy system investigation.
- PyPSA-SPICE-Vis as a visual tool for easy visualization of model outputs.
- Extensive documentation to facilitate working with the model.

1.2 Types of models outside the scope of PyPSA-SPICE

PyPSA-SPICE is not designed for modeling power or energy systems with very high geographic resolution, such as load flow modeling. Instead, it prioritizes ease of building country- or region-level models using manually provided custom data. For this reason, it is best suited for models with a maximum of 10–15 regional nodes.

- For higher geographic detail other open-source PyPSA-based frameworks can be used:
- PyPSA-Eur
- PyPSA-meets-Earth
- PyPSA-SPICE requires total energy demand as an input and does not optimize total demand, modal shifts, or other demand-side dynamics.

1.3 Studies conducted using PyPSA-SPICE

Please refer to Publication to see the publications or studies using the PyPSA-SPICE model builder.

1.4 Visualisation of PyPSA-SPICE Results

To make it easy to visualise and compare scenario outputs, we provide an open-source library PyPSA-SPICE-Vis within the model builder.

1.5 Citing PyPSA-SPICE

Please use the citation below:

- Agora Think Tanks (2025): PyPSA-SPICE: PyPSA-based Sector-Coupled Optimisation for Pathways Exploration

1.6 Contributing

We welcome contributions from anyone interested in improving this project. Please take a moment to review our Contributing Guide and Code of Conduct. If you have ideas, suggestions, or encounter any issues, feel free to open an issue or submit a pull request on GitHub.

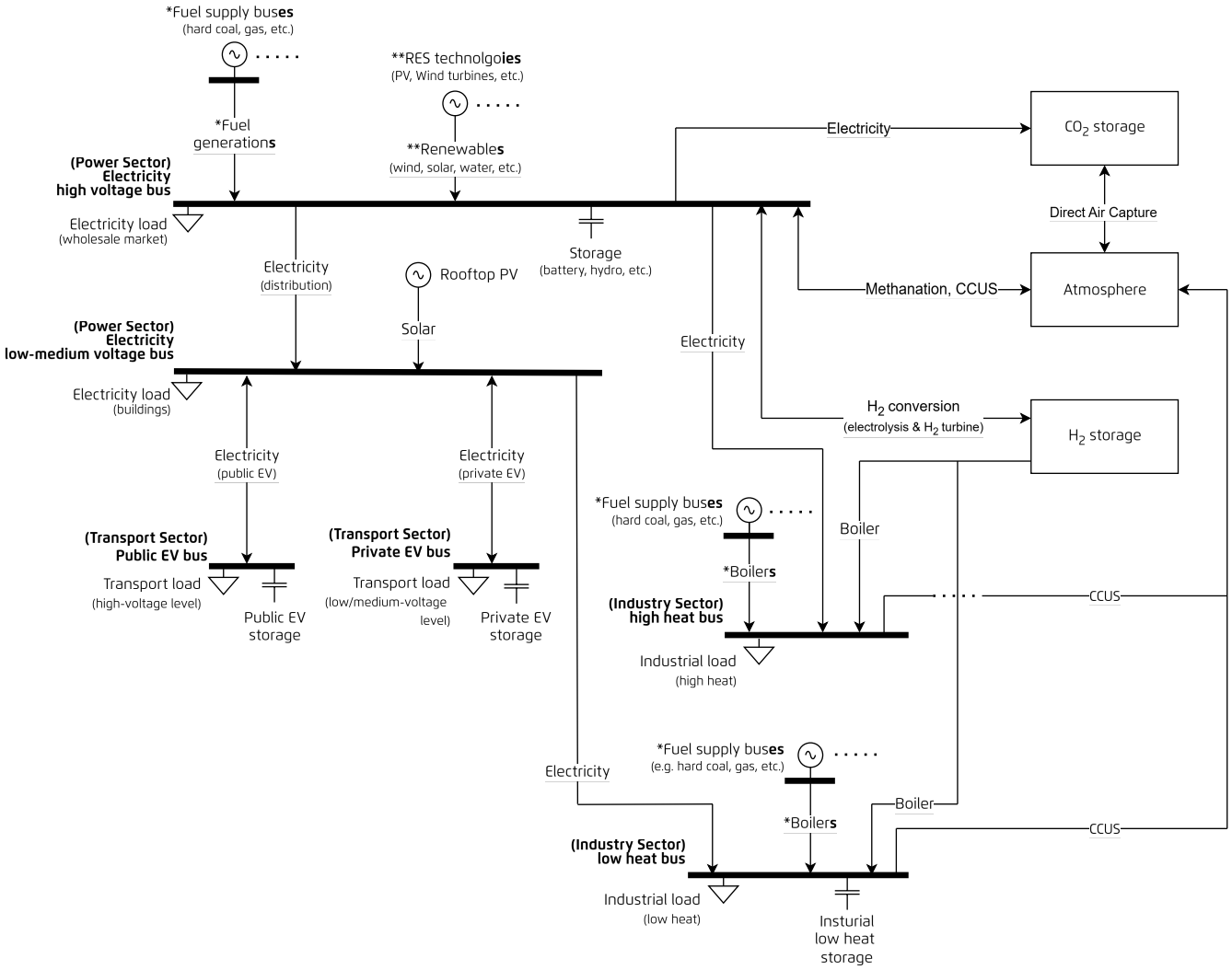
1.7 License

2. 📖 User Guide

2.1 Model Builder Methodology

PyPSA-SPICE is a least-cost optimization model builder designed to evaluate long-term national energy scenarios at a nodal network level. Built on the PyPSA framework, it adopts a multi-sectoral cost optimization approach with a primary focus on the power system.

The diagram below illustrates the energy flows within a single node:



*The diagram displays one bus with one generator and one link as an example to display how the fuel supply hub provides fuels to generate electricity.
In the model, there are multiple sources with this structure to control each fuel supply.

**The diagram displays one generator as an example to display how the renewable sources generate electricity.
In the model, there are multiple sources with this structure to control the generation of each renewable sources.

Each component in this model follows the definitions from PyPSA components and the system is structured into three main sectors:

- Power sector
- Industry sector
- Transport sector

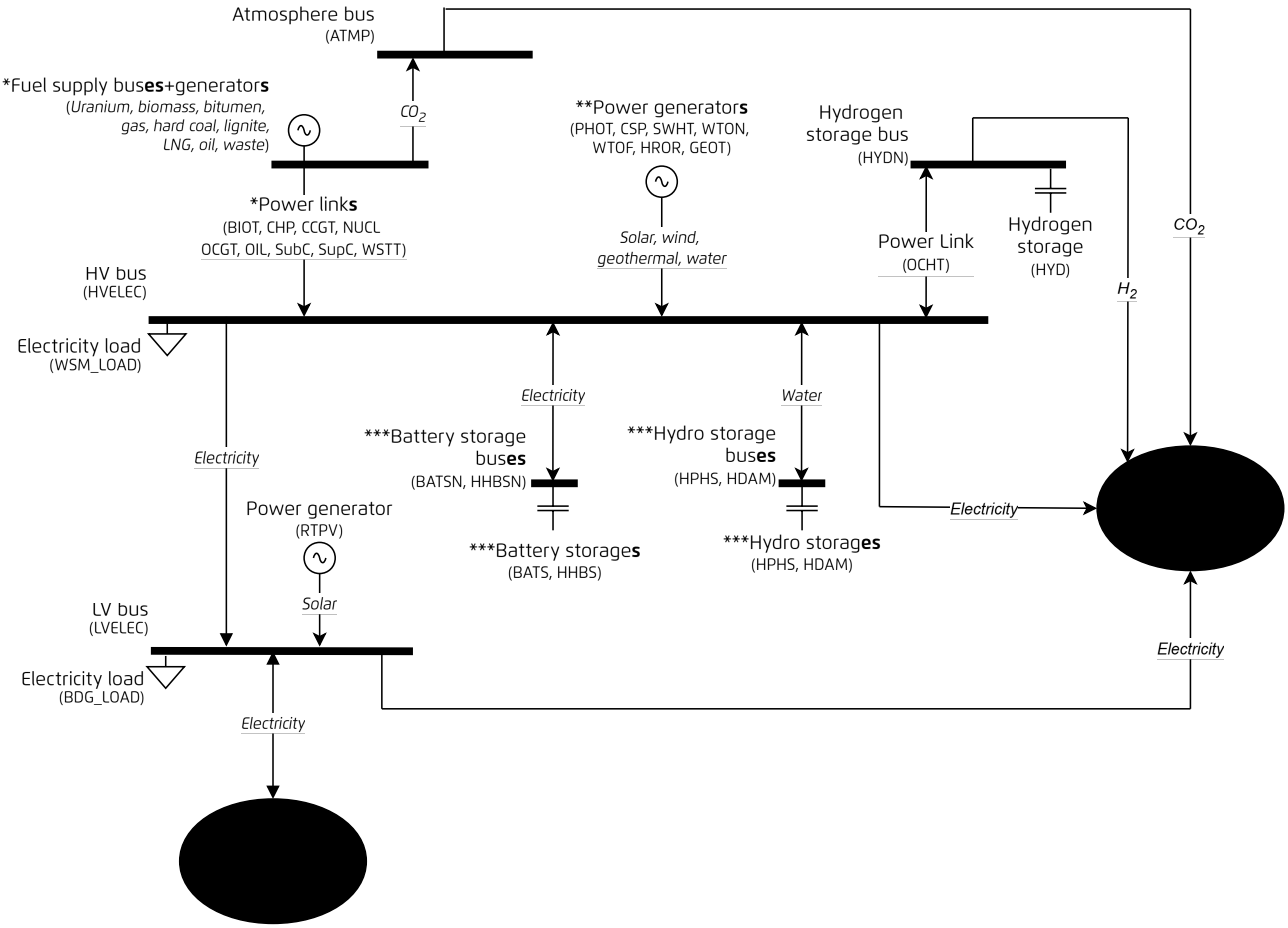
You can find more detailed information within each sector's dedicated section.

2.2 Power Sector

2.2.1 Key Features

- Co-optimisation of generation and capacity expansion including interconnections.
- Myopic (Year-by-year) optimisation. Each year is optimised independently, without assuming knowledge of future developments.
- Brownfield modelling approach. The model builds on existing infrastructure, meaning capacity from previous years is retained and carried forward.

PyPSA-SPICE follows the component definitions from PyPSA Components. The diagram below illustrates all components involved in energy flows at a single node in the power sector.



*The diagram displays one bus with one generator and one link as an example to display how the fuel supply hub provides fuels to generate electricity. In the model, there are multiple sources with this structure to control each fuel supply.

**The diagram displays one generator as an example to display how the renewable sources generate electricity. In the model, there are multiple sources with this structure to control the generation of each renewable sources.

***The diagram displays one bus with a store/storage unit as an example to display the electricity or water storage. In the model, there are multiple sources with this structure to control each storage.

2.2.2 Power Generators

All the listed components are defined as `Generator` in PyPSA.

Abbreviation	Full Name
`CSP`	Concentrated solar power plant
`GEOT`	Geothermal power plant
`HROR`	Hydro run-of-river
`PHOT`	Solar PV
`RTPV`	Rooftop PV
`WTOF`	Offshore wind
`WTON`	Onshore wind

2.2.3 Power Links

All the listed components are defined as `Link` in PyPSA.

Abbreviation	Full Name
`BIOT`	Biomass power plant
`CCGT`	Combined-cycle gas turbine power plant
`CHP`	Combined heat and power plant
`ELTZ`	Electrolyser (for hydrogen production)
`NUCL`	Nuclear power plant
`OCGT`	Open-cycle gas turbine power plant
`OCHT`	Open-cycle hydrogen turbine power plant
`OILT`	Oil turbine power plant power plant
`SubC`	Subcritical coal-fired power plant
`SupC`	Supercritical coal-fired power plant
`WSTT`	Waste-to-Energy power plant

2.2.4 Storage Capacity

The following component is defined as `StorageUnit` in PyPSA.

Storages can be modelled with two approaches.

1. Fixed Energy/Power Ratio: In this case, the energy to power ratio for storage is predefined. You can use multiple storage type with different energy to power ratio. For example, `BATS` with E/P ratio of 4 and `BATS` with E/P ratio of 8 representing different energy to power ratio and the model will optimise the capacity of each of these technology. In this case, PyPSA type `Storage_units` can be used for modelling and defining the energy to power ratio in `Technologies.csv`.
2. Variable Energy/Power Ratio: If you want the model to optimise the energy/power ratio of storage your have to model it using a combination of `links` + `store` component. This requires separate inputs like costs for capacity and energy component of the storage inputs.

> **Tip**
>
> In PyPSA components, `StorageUnit` is modelled as a storage asset with a fixed energy-to-power ratio defined by `max_hours` of the nominal power (you can also refer to PyPSA Components - Storage Unit for more information). Thus, in PyPSA-SPICE model builder, hydro dam `HDAM` is defined as a `StorageUnit` and it is given in storage capcaity only to represent nominal power-related params.
>
> To model the storage energy separately from the power capacity, `store` + 2 `links` is a better combination. You can refer to Storage Energy for more information. Technologies defined in the storage energy require storage capacity if the carrier is related to electricity (power).

Abbreviation	Full Name
`HDAM`	Hydro dam
`BATS`	Utility-scale battery storage
`HHBS`	Household battery storage
`HPHS`	Hydro pumped storage

2.2.5 Storage Energy

All the listed components are defined as `Store` in PyPSA.

> **Tip**
>
> In PyPSA components, `Store` is modelled as a storage asset with only energy storage. It can optimise energy capacity separately from the power capacity with a combination of `store` + 2 `links`. The links represent charging and discharging characteristics to control the power output. Marginal cost and efficiency of charging and discharging can be defined in each link.
>
> In PyPSA-SPICE model builder, technologies that are defined as storage energy, they should also be included in Storage Capacity to describe charging and discharging processes. The links are created automatically , and hence it's not required to add charging and discharging links inside Power Links.
>
> Detailed information and example can be found in PyPSA Components - Store and Replace StorageUnits with fundamental Links and Stores.

Abbreviation	Full Name
`CO2STOR`	CO_2 storage
`BATS`	Utility-scale battery storage
`HHBS`	Household battery storage
`HPHS`	Hydro pumped storage

2.2.6 Carriers

Abbreviation	Full Name
`Bio`	Biomass
`Bit`	Bituminous or brown coal
`CO2`	Carbon dioxide (in the atmosphere)
`Co2stor`	Captured carbon dioxide
`Electricity`	Electricity
`Gas`	Domestic natural gas
`Gas-imp`	Imported natural gas
`High_Heat`	High-temperature heat (> 350°C)
`Hrdc`	Anthracite or hard coal
`Hyd`	Hydrogen
`Lig`	Lignite
`Lng`	Liquefied natural gas
`Low_Heat`	Low/Medium-temperature heat (< 350°C)
`Oil`	Oil
`Uranium`	Uranium
`Waste`	Waste

2.2.7 Buses

Abbreviation	Full Name
`ATMP`	Atmosphere
`BATSN`	Lithium battery storage
`BION`	Biomass
`BITN`	Bituminous
`CO2STORN`	CO_2 storage
`GASN`	Gas
`HHBSN`	Household battery storage
`HPHSN`	Hydro pumped storage
`HRDCN`	Anthracite or hard coal
`HVELEC`	High-voltage electricity
`HYDN`	Hydrogen
`LIGN`	Lignite
`LNGN`	liquefied natural gas
`LVELEC`	Low-voltage electricity
`NUCLN`	Uranium
`OILN`	Oil
`WSTN`	Waste

2.2.8 Other Components

Abbreviation	Full Name
`co2Price`	Price of emitting one unit of CO_2 into the atmosphere
`r`	Interest rate
`HV_LOAD`	Wholesale market load (High voltage level)
`LV_LOAD`	Building load (low/medium voltage level)

2.2.9 Custom Constraints (Defined in the `config.yaml` File)

- CO2 management
- Energy independence
- Fuel production constraint
- Reserve margin
- Renewable generation share constraint
- Must run constraint of thermal generators
- Capacity factor constraint

You can refer to Model Builder Constraints for more information.

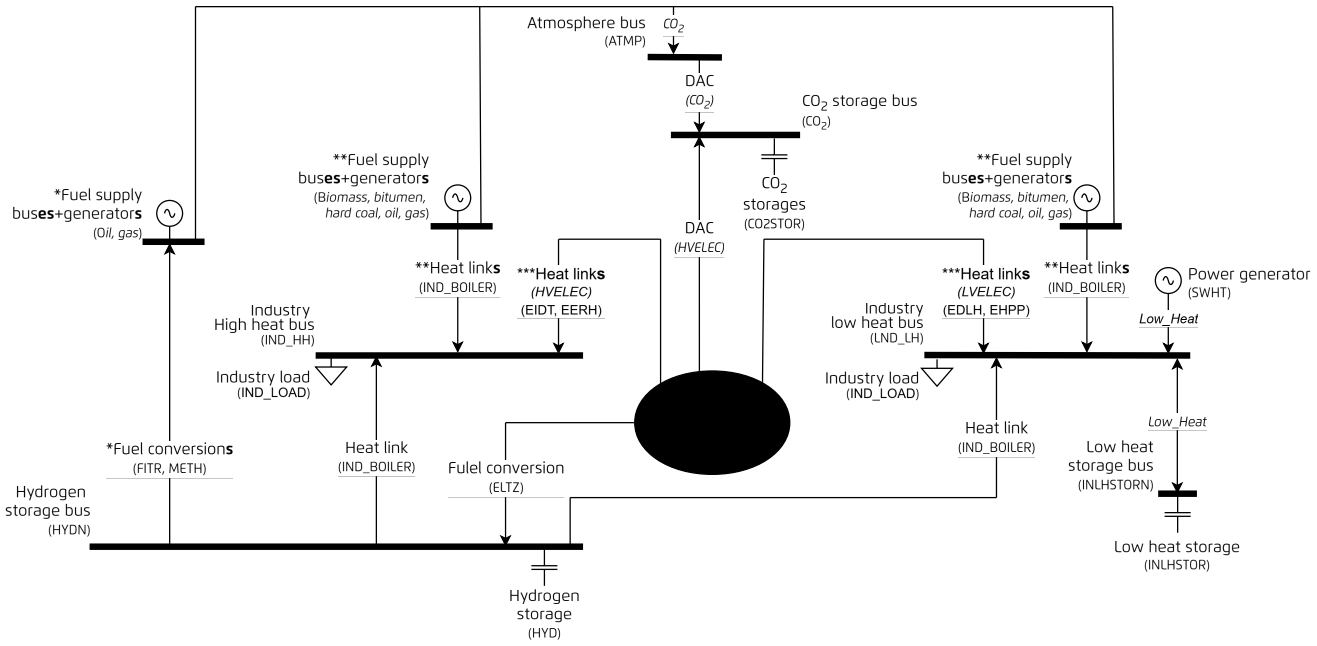
2.3 Industry Sector

2.3.1 Key Features

Optimisation of industry heat supply at two different temperature levels:

- High-temperature (above 350°C)
- Low/medium-temperature (350°C or below)

The structure and functionality of components follow the PyPSA Components. The diagram below shows the full set of components involved in energy flows for a single industrial node.



*The diagram displays one bus with one generator and one link as an example to display the fuel conversion.
In the model, there are multiple sources with this structure to control each fuel conversion process.

**The diagram displays one bus with one generator and one link as an example to display how the boilers generate heat.
In the model, there are multiple sources with this structure to control each boiler.

***The diagram displays one link as an example to display how heat is converted from the power sector.
In the model, there are multiple sources with this structure to control each heat link.

2.3.2 Heat Generators

The following component is defined as `Generator` in PyPSA.

Abbreviation	Full Name
`SWHT`	Solar hot water heater

2.3.3 Heat Links

All the listed components are defined as `Link` in PyPSA.

Abbreviation	Full Name
`EERH`	Electric resistance heater
`EDLH`	Dielectric heating technology
`EHPP`	Industry heat pump
`EIDT`	Induction heat boiler
`FITR`	Fischer-Tropsch process
`IND_BOILER`	Industry heat boiler
`INLHSTOR`	Low-temperature heat Storage
`METH`	Methanation

2.3.4 Fuel Conversion

All the listed components are defined as `Link` in PyPSA.

Abbreviation	Full Name
`ELTZ`	Electrolyser (for hydrogen production)
`FITR`	Fischer-Tropsch process

2.3.5 Direct Air Capture

The following components is defined as `Link` in PyPSA.

Abbreviation	Full Name
`DAC`	Direct air capture

2.3.6 Storage Capacity

The following component is defined as `StorageUnit` in PyPSA.

> **Tip**
>
> In PyPSA components, `StorageUnit` is modelled as a storage asset with a fixed energy-to-power ratio defined by `max_hours` of the nominal power (you can also refer to PyPSA Components - Storage Unit for more information).
>
> To model the storage energy separately from the power capacity, `store` + 2 `links` is a better combination. You can refer to Storage Energy for more information. Technologies defined in the storage energy require storage capacity if the carrier is related to electricity (power).

Abbreviation	Full Name
`INLHSTOR`	Low-temperature heat storage

2.3.7 Storage Energy

All the listed components are defined as `Store` in PyPSA.

> **Tip**
>
> In PyPSA components, `Store` is modelled as a storage asset with only energy storage. It can optimise energy capacity separately from the power capacity with a combination of `store` + 2 `links`. The links represent charging and discharging characteristics to control the power output. Marginal cost and efficiency of charging and discharging can be defined in each link.
>
> IIn the industry sector of PyPSA-SPICE model builder, the media `electricity` is replaced by `low temperature heat` in the storage process. Technologies that are defined as storage energy, they should also be included in Storage Capacity to describe charging and discharging processes. The links are created automatically , and hence it's not required to add charging and discharging links inside Heat Links, Fuel Conversion, or Direct Air Capture.
>
> Detailed information and example can be found in PyPSA Components - Store and Replace StorageUnits with fundamental Links and Stores.

Abbreviation	Full Name
`INLHSTOR`	Low-temperature heat storage

2.3.8 Carriers

Abbreviation	Full Name
`Bio`	Biomass
`CO2`	Carbon dioxide (in the atmosphere)
`Co2stor`	Captured carbon dioxide
`Electricity`	Electricity
`Gas`	Domestic natural gas
`High_Heat`	High-temperature heat (> 350°C)
`Hrdc`	Anthracite or hard coal
`Hyd`	Hydrogen
`Low_Heat`	Low/Medium-temperature heat (< 350°C)
`Oil`	Oil

2.3.9 Buses

Abbreviation	Full Name
`CO2STORN`	CO_2 storage
`HVELEC`	High-voltage electricity
`HYDN`	Hydrogen
`LVELEC`	Low-voltage electricity
`IND_LH`	Industrial low-temperature heat
`IND_HH`	Industrial high-temperature heat
`INLHSTORN`	Industrial low-temperature heat storage

2.3.10 Other Components

Abbreviation	Full Name
`co2Price`	Price of emitting one unit of CO_2 into the atmosphere
`IND_LOAD`	Industrial load (both high- and low-temperature heat)

2.3.11 Custom Constraints (Defined in the `config.yaml` File)

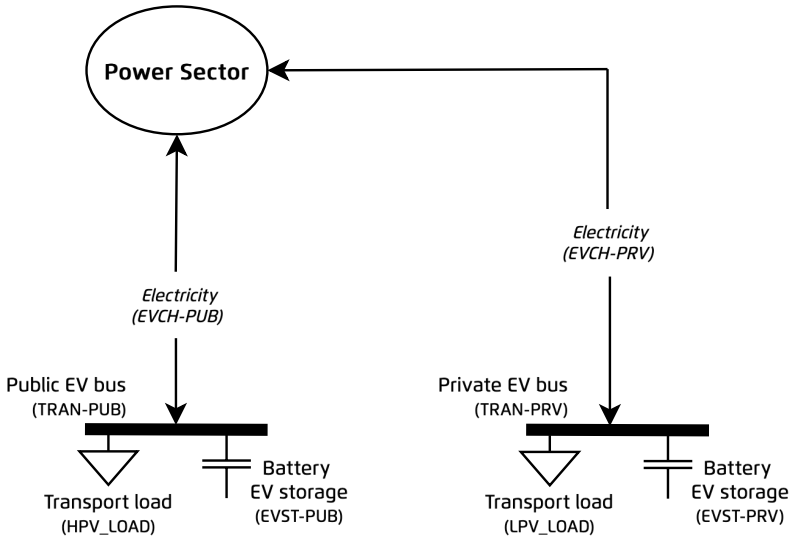
- Coming soon...

2.4 Transport Sector

2.4.1 Key Features

- Optimal charging of electric vehicles (EVs), taking into account availability and charging constraints.
- Supply of fuels for transport.

The structure and function of each component follow the definitions from the PyPSA components. The diagram below illustrates the components and energy flows at a single node in the transport sector.



> Tip
>
> In the transport sector, PyPSA-SPICE separated the demand and energy flow into two categories: private and public sector. The public sector (`PUB`) encompasses public transportation services, while the private sector (`PRV`) refers to personally owned vehicles.

2.4.2 Electric Vehicle Chargers

All the listed components are defined as `Link` in PyPSA.

Abbreviation	Full Name
`EVCH-PRV`	Electric vehicle charger (Private)
`EVCH-PUB`	Electric vehicle charger (Public)

2.4.3 Electric Vehicle Storage

All the listed components are defined as `Store` in PyPSA.

> **Tip**
>
> In PyPSA components, `Store` is modelled as a storage asset with only energy storage. It can optimise energy capacity separately from the power capacity with a combination of `store` + 2 `links`. The links represent charging and discharging characteristics to control the power output. Marginal cost and efficiency of charging and discharging can be defined in each link.
>
> In the transport sector of PyPSA-SPICE model builder, technologies that are defined as storage energy, their links of charging and discharging links are defined in Electric Vehicle Chargers.
>
> Detailed information and example can be found in PyPSA Components - Store and Replace StorageUnits with fundamental Links and Stores.

Abbreviation	Full Name
`EVST-PRV`	Electric vehicle storage (Private)
`EVST-PUB`	Electric vehicle storage (Public)

2.4.4 Carriers

Abbreviation	Full Name
`Electricity`	Electricity

2.4.5 Buses

Abbreviation	Full Name
`HVELEC`	High-voltage electricity
`LVELEC`	Low-voltage electricity
`TRAN-PUB`	Public electric vehicle
`TRAN-PRV`	Private electric vehicle

2.4.6 Other Components

Abbreviation	Full Name
`HPV_LOAD`	Transport load (High voltage level)
`LPV_LOAD`	Transport load (low/medium voltage level)

2.4.7 Custom Constraints (Defined in the `config.yaml` File)

- Coming soon...

2.5 Standard Output from the Snakemake Workflow

The following NetCDF files, CSVs, and charts are generated automatically after the whole Snakemake workflow is successfully executed.

2.5.1 NetCDF Files (`*.nc` Files)

NetCDF (Network Common Data Form) is a data format for efficiently storing multi-dimensional arrays. By default, the PyPSA-SPICE model creates:

- Pre-solve networks in the `results/pre-solve` directory.
- Pre-solve-brownfield networks in the `results/pre-solve-brownfield` directory.
- Post networks in the `results/post-solve` directory.

These `*.nc` files allow users to examine the model's structure, inputs, and results before and after optimisation.

2.5.2 Excel-Ready CSVs

The following CSV files are created automatically. Each file represents a key indicator, broken down by sector and modeling year or hour, depending on granularity.

File Name	Sector	Unit	Description
`ene_avg_fuel_costs_fuel_yearly`	Energy	currency/MWh_{th}	Average fuel costs by modeling year
`ene_costs_opex_capex_yearly`	Energy	million currency	Total CAPEX and OPEX in energy sectors by modeling year
`ene_dmd_by_carrier_yearly`	Energy	TWh	Energy demand by carrier (e.g., `Electricity`) by modeling year
`ene_emi_by_carrier_by_sector_yearly`	Energy	$MtCO_2$	Total emissions by carrier and sector by modeling year
`ene_fom_by_type_yearly`	Energy	million currency	Total fixed operation and maintenance cost by technology (e.g., `CCGT`) by modeling year
`ene_gen_by_carrier_yearly`	Energy	TWh	Total generation by carrier (e.g., `Electricity`) by modeling year
`ene_opex_by_type_yearly`	Energy	million currency	OPEX in energy sectors by technology (e.g., `CCGT`) by modeling year
`ind_cap_by_carrier_by_region_yearly`	Industry	GW	Installed capacity in the industry sector by carrier and region/node, and by modeling year
`ind_cap_by_type_by_carrier_yearly`	Industry	GW	Installed capacity in the industry sector by technology and carrier, and by modeling year
`ind_emi_by_carrier_yearly`	Industry	$MtCO_2$	Emissions in the industry sector by carrier by modeling year
`ind_gen_by_carrier_by_region_yearly`	Industry	TWh	Generation in the industry sector by carrier and region/node, and by modeling year
`ind_gen_by_type_by_carrier_by_heatgroup_yearly`	Industry	TWh	Generation in the industry sector by technology and carrier, heating group, and by modeling year
`ind_gen_by_type_hourly`	Industry	MW	Generation in the industry sector by technology by modeling year
`ind_hh_gen_by_type_hourly`	Industry	MW	Generation in the industry sector (high-heat) by technology by modeling year
`ind_lh_gen_by_type_hourly`	Industry	MW	Generation in the industry sector (low-heat) by technology by modeling year
`pow_bats_charging_hourly`	Power	MW	Hourly battery charging profile by modeling year

File Name	Sector	Unit	Description
`pow_bats_ep_ratio`	Power	-	Battery Energy-to-Power ratio
`pow_battery_flows_by_region_hourly`	Power	MW	Hourly battery flows between different regions/nodes by modeling year
`pow_cap_by_region_yearly`	Power	GW	Installed capacity in the power sector by region/node by modeling year
`pow_cap_by_type_yearly`	Power	GW	Installed capacity in the power sector by technology (e.g., `CCGT`) by modeling year
`pow_cap_by_type_by_region_yearly`	Power	GW	Installed capacity in the power sector by technology (e.g., `CCGT`) by region/node by modeling year
`pow_capex_by_type_yearly`	Power	million currency	CAPEX in the power sector by technology (e.g., `CCGT`) by modeling year
`pow_elec_load_by_sector_hourly`	Power	MW	Hourly electricity load by sector
`pow_emi_by_carrier_yearly`	Power	$MtCO_2$	Emissions in the power sector by carrier (e.g., `Gas`) by modeling year
`pow_flh_by_type_yearly`	Power	Hours	Full load hours by technology (e.g., `CCGT`) by modeling year
`pow_fom_by_type_yearly`	Power	million currency	Fixed operation and maintenance cost in the power sector by technology (e.g., `CCGT`) by modeling year
`pow_gen_by_category_share_yearly`	Power	%	Fossil-Renewables share by modeling years
`pow_gen_by_type_hourly`	Power	MW	Hourly generation in the power sector by technology (e.g., `CCGT`)
`pow_gen_by_type_region_hourly`	Power	TWh	Hourly generation in the power sector by region/node by modeling year
`pow_gen_by_type_yearly`	Power	TWh	Generation in the power sector by technology (e.g., `CCGT`) by modeling year
`pow_hdam_flows_by_region_hourly`	Power	MW	Hourly hydro dam flow profile by modeling year
`pow_hphs_flows_by_region_hourly`	Power	MW	Hourly hydro pumped storage profile by modeling year
`pow_inter_cf_by_region_yearly`	Power	-	Capacity factor of the interconnectors by region/node by modeling year
`pow_intercap_by_region_yearly`	Power	GW	

File Name	Sector	Unit	Description
			Installed capacity of the interconnectors by region/node by modeling year
`pow_marginal_price_by_region_hourly`	Power	currency/MWh	Hourly marginal price by region/node by modeling year
`pow_nodal_flow_hourly`	Power	MW	Hourly exchange flow between different regions/nodes by modeling year
`pow_opex_by_type_yearly`	Power	million currency	OPEX in the power sector by technology (e.g., `CCGT`) by modeling year
`pow_overnight_inv_by_type_yearly`	Power	million currency	Overnight investment cost in the power sector by technology (e.g., `CCGT`) by modeling year
`pow_reserve_by_type_hourly`	Power	TWh	Reserve by technology (e.g., onshore wind) by modeling year
`tran_capex_by_type_yearly`	Transport	million currency	CAPEX in the transport sector by technology by modeling year
`tran_charger_capacity_by_region_yearly`	Transport	GW	Capacity of the chargers in the transport sector by region by modeling year
`tran_charger_capacity_by_type_yearly`	Transport	GW	Capacity of the chargers in the transport sector by technology by modeling year
`tran_load_charging_all_hourly_by_region`	Transport	MW	Hourly charging/discharging/load status in the transport sector by region/node by modeling year
`tran_load_charging_all_hourly_by_type`	Transport	MW	Hourly charging/discharging/load status in the transport sector by technology by modeling year
`tran_load_charging_all_hourly`	Transport	MW	Hourly charging/discharging/load status in the transport sector by modeling year
`tran_storage_capacity_by_region_yearly`	Transport	GW	Capacity of the storage in the transport sector by region by modeling year
`tran_storage_capacity_by_type_yearly`	Transport	GW	Capacity of the storage in the transport sector by technology by modeling year

2.5.3 Jupyter Notebook

We provide a `post-analysis/post_analysis.ipynb` for manual exploration of the pre-solve, pre-solve-brownfield, or post-solve network files.

2.5.4 Dynamic Visualization

To make it easier to explore and compare model outputs, we provide an open-source library PyPSA-SPICE-Vis, an interactive visualisation library that generates dynamic charts using the outputs listed above.

2.6 Troubleshooting

2.6.1 Setup Debugger

To test the workflow using Snakemake rules, a configuration file `launch.json` with the following content is required:

Debugger configurations

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "mock snakemake",
            "type": "python",
            "request": "launch",
            "program": "${file}",
            "console": "integratedTerminal",
            "justMyCode": false,
            "cwd": "${cwd}${pathSeparator}scripts"
        }
    ]
}
```

Once this file is set up, you can easily run any Python file in the `scripts` folder under debugger mode for testing.

2.6.2 Infeasibility Issues

If your model is infeasible or unbounded, it often means that your input settings are leading to a situation where PyPSA can't solve one or more objective functions. Common causes are:

- Insufficient generator capacity at a bus to meet the load across all snapshots.
- Must-run generators (`p_min_pu`) produce more power than the load at certain snapshots, causing excess power (i.e., power dumping).
- Negative values assigned to capacity parameters.
- Maximum capacity (`p_nom_max`) is smaller than minimum capacity (`p_nom_min`).
- Storage units has inflow and cyclic charging (`cyclic_state_of_charge = True`) but no discharging capability.

2.6.3 Tips for Diagnosing the Problem

Try the following steps to identify and fix the issue:

- Run `pypsa.network.consistency_check()` to check if any warnings appear.
- Temporarily disable custom features or user extensions to isolate the cause.
- Ensure load-shedding generators are added to every bus.
- Check `pypsa.network.generators.p_min_pu` to identify any must-run generators. Then verify if their generation exceeds the corresponding load in `pypsa.network.loads_t.p_set` .
- Compare the `p_nom_max` and `p_nom_min` values for each component to ensure they make sense (i.e., max ≥ min).
- Try disabling cyclic charging for storage units: `pypsa.network.storage_units.cyclic_state_of_charge = False` .
- Double-check for any negative values in:
    - `pypsa.network.{component}.p_nom`
    - `pypsa.network.{component}.p_nom_max`
    - `pypsa.network.{component}.p_nom_min`

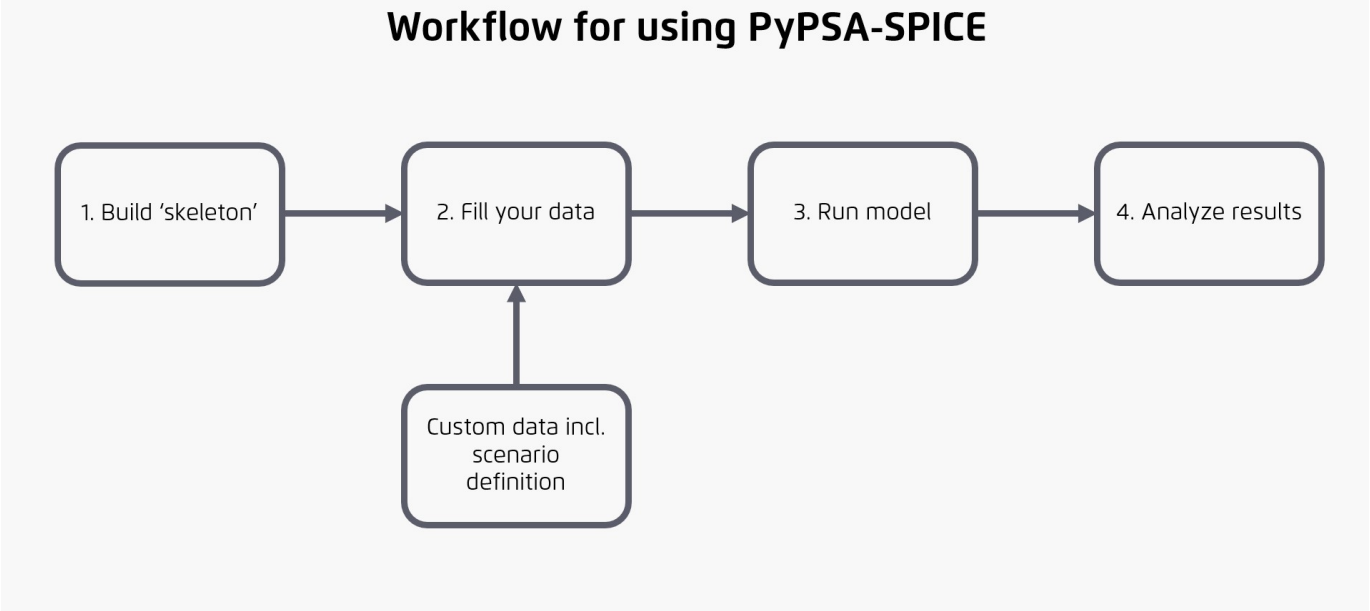
2.6.4 More detailedinformation

PyPSA also provides official guidance on solving infeasiblity issues. You can also explore the link to seek for a good solution.

3. 🚀 Getting Started

3.1 Overview of workflow

The diagram below outlines the process for preparing and entering input data, running the model, and performing post-analysis along with visualization. The following sections provide additional details and guidance for each step.



> **Info**
>
> If you are considering using this model builder, please reach out to us at Email_to_be_added. We would be happy to help you get started. If you encounter a bug, please create a new issue. For new ideas or feature requests, you can start a conversation in the discussions section of the repository.

1. Build skeleton: This is the initial step where you define the scope and resolution of the model and run a script to generate empty data files. At this stage, it is important to specify the regions must be included in the model

2. Fill data: Use the generated empty CSV files to input your custom data. You can also create multiple scenarios with distinct data and conditions. We recommend maintaining a separate repository for storing and modifying your input data.

3. Run model: Execute the model with your defined scenarios via the command line.

4. Analyse results: Evaluate the model output using either Jupyter Notebook or locally run interactive app we provide - PyPSA-SPICE-Vis.

3.2 Installation

3.2.1 Clone the Repository

First, clone the PyPSA-SPICE repository using the Git version control system. Important: the path to the directory where the repository is cloned must not contain any spaces.

If Git is not installed on your system, please follow the Git installation instructions.

Cloning the repository

```
git clone https://github.com/agoenergy/pypsa-spice.git
cd pypsa-spice
```

3.2.2 Install Python Dependencies

PyPSA-SPICE needs a set of Python packages to function. We recommend using Conda, a package and environment management system, to handle these dependencies.

Start by installing Miniconda, a lightweight version of Anaconda that includes only Conda and its core dependencies. If you already have Conda installed, you can skip this step. For installation instructions tailored to your operating system, refer to the official Conda installation guide.

Once Conda is installed, we recommend installing Mamba, a fast drop-in replacement for Conda that significantly speeds up environment installation. According to the official mamba documentation, the recommended way to install it is via Miniforge3, a minimal Conda installer bundled with Mamba.

Installation steps for Miniforge3 vary depending on your operating system. You can find platform-specific instructions in the official Miniforge guide.

For example, if you're using Linux system, you can install Miniforge3 by running the following command outside the PyPSA-SPICE folder and following the prompts:

Installing Miniforge3 on Linux

```
cd ..
curl -L -O "https://github.com/conda-forge/miniforge/releases/latest/download/Miniforge3-Linux-x86_64.sh"
bash Miniforge3-$ Miniforge3-Linux-x86_64.sh
```

You can switch to any other directory outside the folder you cloned PyPSA-SPICE.

Important Notes:

1. Mambaforge is deprecated as of July 2024 and was officially retired after January 2025. For this reason, we recommend using Miniforge3 instead.
2. Install Miniforge3 or mamba packages only in the base Conda environment. Installing them in other environments may lead to compatibility issues or unexpected errors.
3. Miniforge3 and Mambaforge use different environment paths. If you switch from one to the other, you will need to recreate all Conda environments, as they are not shared between the two setups.

The required Python packages for PyPSA-SPICE are listed in the `environment.yaml` (via conda or mamba) and `requirements.txt` (via pip). You can create and activate the environment (which is called `pypsa-spice`) using the following commands:

Installing and activating the virtual environment

```
mamba env create -f envs/environment.yaml
conda activate pypsa-spice
```

Note that the environment activation is local to the currently open terminal session. If you open a new terminal window, you will need to re-run the activation command.

3.2.3 Install a Solver

Th network model created in PyPSA-SPICE model builder is passed to an external solver to perform total annual system cost minimization and obtain optimal power flows. PyPSA-SPICE is compatible with several solvers that can be installed via Python. In our default environment, Gurobi and HiGHs solvers are installed (packages installed not licenses). Below is a list of supported solvers along with links to their official installation guides for different operating systems:

Solver	License type	Installation guide
Ipopt	Free & open-source	Ipopt
Cbc	Free & open-source	Cbc
GLPK	Free & open-source	GLPK /WinGLPK
HiGHs	Free & open-source	highspy
Gurobi	commercial	gurobipy
CPLEX	commercial	CPLEX
FICO® Xpress Solver	commercial	FICO® Xpress Solver

Commercial solvers such as Gurobi currently significantly outperform open-source solvers for large-scale problems. In some cases, using a commercial solver may be necessary to obtain a feasible solution. However, open-source solvers are being improved recently, and some of them already perform very well for small to medium-sized problems. Please check the solver benchmarking results to compare their performance.

on Mac or Linux **on Windows**

```
conda activate pypsa-spice
conda install -c conda-forge ipopt coincbc

conda activate pypsa-spice
conda install -c conda-forge ipopt glpk
```

On Windows, new versions of `ipopt` have caused problems. Consider downgrading to version 3.11.1.

3.2.4 Set up the Default Configuration

PyPSA-SPICE requires several configuration options to be defined in a `config.yaml` file located in the root directory of the project. An example file, `config.default.yaml`, is provided in the repository as a reference.

Before running PyPSA-SPICE for the first time, create your own `config.yaml` by copying the example file:

Creating the config.yaml file

```
cp config.default.yaml config.yaml
```

Users are encouraged to regularly compare their own `config.yaml` with `config.default.yaml` when pulling updates from the remote repository, to ensure compatibility with any new or changed configuration options.

3.2.5 Quick Execution of the Model Builder using Template Data

To have a first glance of how the model builder works, template data in `/root/data/global_csv_templates` folder can be used. After completing the installation, run the entire workflow with the following command:

Running the entire workflow using this single command

```
snakemake -j1 -c4 solve_all_networks # (1)!
```

1. `-j1` means running only 1 job in parallel at a time and `-c4` means allowing each job to use up to 4 CPU threads.

or

Running the entire workflow using this call command

```
snakemake -call
```

For more information, please refer to the Snakemake documentation to adjust the cores and threads to use.

3.3 Input Data

3.3.1 Input Data: Define a New Model

This section explains how to set up a new model for a particular country/region using the PyPSA-SPICE model builder. Once your model is step you can run the model as described in model execution.

Steps of setting up a new model:

1. Set up the configuration file (`config.yaml`).
2. Run `python script/build_skeleton.py` to create a folder structure and template CSV files for your input data.
3. Fill in the skeleton CSVs with the required data manually or using available resources.

An example structure created by `build_skeleton` is displayed below. The following sections will use this example to explain the settings.

Step 1: Set up the Configuration File

Setting up the config requires defining the scope and resolution of the model. Specifically, defining which countries/regions will be represented in the model and for which year the model will be run. While it is possible to change these after initial model is created, it would require significant effort to add new regions/years in the input CSVs.

The `config.yaml` contains two parts:

- Skeleton settings: for defining the overall structure of the model that is used to create the input data files.
- Model settings: for model execution (see Model Configuration) and turning on/off different conditions.

In the first part (named Init settings), following parameters should be defined as shown below.

Init settings in the config.yaml file

```yaml
path_configs: #(1)!
  input_dir: data/example/
  results_dir: results/
  project_name: project_01
  scenario_name: scenario_01 # (2)!

base_configs:
  countries: ["XY", "YZ"] # (3)!
  regions:
    XY: ["NR","CE", "SO"] # (4)!
    YZ: ["NR","CE", "SO"]
  years: [2025, 2030, 2035, 2040, 2045, 2050] # (5)!
  sector: ["p-i-t"] # (6)!
```

1. This section is for configuring directory structure for storing model inputs and results.
2. A custom name you define for the scenario in your model.
3. A list containing 2-letter country codes according to ISO 3166.
4. List of subregions in each country.
5. Modelled years should be provided as a list.
6. Options: [`p` , `p-i` , `p-t` , `p-i-t`], representing power (`p`), industry (`i`), and transport (`t`) sectors.

The final skeleton folder path will follow this structure:: `input_dir` / `countries` / `scenario_name` .

By setting different `scenario_name` and country or regional settings in the `base_configs` section (see details in Model Configuration), a new skeleton structure under the same `input_dir` folder will be created.

Step 2: Build the Skeleton

After modifying the configuration file, run the following command in your terminal.

Generating the skeleton folder

```
python script/build_skeleton.py
```

This step creates your skeleton folder and files which can be feed with your data.

Structure of Folder and files created by build skeleton script

```
📦 data
 ├ 🗁 example
 |  └ 🗁 project_01
 |   └ 🗁 scenario_01
 |    ├ 🗁 industry
 |    |  ├ 🗎 buses.csv
 |    |  ├ 🗎 decommission_capacity.csv
 |    |  ├ 🗎 direct_air_capture.csv
 |    |  ├ 🗎 fuel_conversion.csv
 |    |  ├ 🗎 heat_generators.csv
 |    |  ├ 🗎 heat_links.csv
 |    |  ├ 🗎 loads.csv
 |    |  ├ 🗎 storage_capacity.csv
 |    |  └ 🗎 storage_energy.csv
 |    ├ 🗁 Power
 |    |  ├ 🗎 buses.csv
 |    |  ├ 🗎 decommission_capacity.csv
 |    |  ├ 🗎 fuel_suppliers.csv
 |    |  ├ 🗎 interconnector.csv
 |    |  ├ 🗎 loads.csv
 |    |  ├ 🗎 power_generators.csv
 |    |  ├ 🗎 storage_capacity.csv
 |    |  ├ 🗎 power_links.csv
 |    |  └ 🗎 storage_energy.csv
 |    └ 🗁 Transport
 |     ├ 🗎 buses.csv
 |     ├ 🗎 loads.csv
 |     ├ 🗎 pev_chargers.csv
 |     └ 🗎 pev_storages.csv
 ├ 🗁 global_csv_templates
 | ├ 🗎 Technologies.csv
 | ├ 🗎 Availability.csv
 | ├ 🗎 Demand_Profile.csv
 | ├ 🗎 PowerPlant_costs.csv
 | ├ 🗎 Renewables_technical_potential.csv
 | ├ 🗎 Storage-Inflows.csv
 | └ 🗎 Storage_costs.csv
 └ 🗁 override_component_attrs
```

Note: The `override_component_attrs` folder supports multi-output for PyPSA components. These files are used internally and don't need to be edited.

> **Tip**
>
> You will only need to run `build_skeleton` ones while setting up the model. After this initial run you should turn of `build_skeleton` in `config.yaml`. Otherwise, the `build_skeleton` will overwrite your data from in the input files and create new empty files. Once you have created a skeleton for one of the scenarios, you can simply copy this folder to create additional scenarios. We recommend to do this after you have filled the data for the first scenario.

Step 3: Fill in the Skeleton CSVs

Once a new skeleton folder is created, project-specific CSV templates will be setup. Each CSV will include placeholders marked with `Please fill here`. These need to be completed with relevant data so the model can perform more accurate optimizations.

To help you fill these files:

- See Regional CSV Template for detailed file descriptions.
- Check Global CSV Template for default global values. If needed, you can copy values from global templates into the regional ones. However, these values are based on the average assumption of global average, and might not be as accurate as valued obtained from regional sources. Global data is useful as a fallback, but using regional data is strongly recommended for more accurate results.

Once all the necessary input data is provided, adjust model and solver settings in Model Configuration and follow Model Execution to understand the model logic and how to run the model.

> **Tip**
>
> By detault, the input structure considers large number of technologies represented in the model. If the particular technologies are not needed in your model, it is good practice to remove the input data for these technologies. You can also define your own technologies and customise the model accordingly.

3.3.2 Input Data: Global CSV Template

Global csvs contain parameters that are typically kept constant accross scenarios. This is to maintain comparability of the scenarios.

Structure of the global CSV template files

```
data
└ global_csv_templates
  ├ Technologies.csv
  ├ Availability.csv
  ├ Demand_Profile.csv
  ├ PowerPlant_costs.csv
  ├ Renewables_technical_potential.csv
  ├ Storage_costs.csv
  └ Storage-Inflows.csv
```

> **Tip**
>
> The currency of all example data is `USD` defined in the `base_configs` section of `config.yaml`. You can refer to Model Builder Configuration for more information.

Technologies

Technologies.csv defines typical technical parameters for each technology used in the model builder. It provides values like efficiency, ramp limits, and availability of various technologies. To add a new technology, insert a new row and fill in all required parameters.

Description of all technical parameters:

Parameter	definition
`country`	2-letter country codes according to ISO 3166 format
`technology`	Abbreviations of the technology(=technologies)
`technology_nomenclature`	Full names of the technology (=technologies)
`carrier`	Resources used by the technologies
`class`	Component class as defined in PyPSA
`efficiency`	Energy conversion efficiency from primary energy to electricity for `Generators`, and to another form of energy for `Links`. For `StorageUnits`, this is the discharge efficiency.
`efficiency2`	Positive values represent emission factor and negative values correspond to the efficiency of generating the second product in a plant
`efficiency3`	Carbon capture efficiency (for CCS technologies)
`efficiency_store`	Efficiency of charging energy into storage
`max_hours`	Maximum charge duration in hours (total storage volume / capacity)
`cyclic_state_of_charge`	If True, the final state of charge equals the initial state of charge
`state_of_charge_initial`	Initial state of charge in MWh before the snapshots in the optimal Power Flow (MWh)
`p_max_pu`	The maximum availability per snapshot per unit of `p_nom`
`p_min_pu`	The minimum availability per snapshot per unit of `p_nom`
`ramp_limit_down`	Maximum active power decrease from one snapshot to the next (per unit)
`ramp_limit_up`	Maximum active power increase from one snapshot to the next (per unit)
`standing_loss`	Hourly energy loss from storage
`r_rating`	Contribution of reserve rating (if used)

Availability

`Availability.csv` contains time-series availability data, mainly for renewable plants. By default, availability is matched using renewables type (e.g., solar photovoltaic (`PHOT`), onshore wind (`WTON`), etc.) and their locations (e.g., region `XY_NO` in country `XY` , region `YZ_SO` in country `YZ`).

If a technology shares the same profile across the country (e.g., electric vehicle charger public (`EVCH-PUB`)), then both region and country fields use the same name (e.g., region `XY` in country `XY`). If the technology is listed and it requires availability profile, but the profile is not in this csv, then it will be defined as constant 1 for all hours.

Demand Profile

`Demand_Profile.csv` stores normalized hourly load profiles, which are scaled using total annual load values of each year to create time-series demand data.

By default, Load profiles are matched based on:

- Profile type and location for power sector loads such as wholesale market load (`HV_LOAD`) and building load (`LV_LOAD`).
- Profile type only for all other loads.

To add new load profiles (e.g., for a new project or country), insert a new row.

Power Plant Costs

`PowerPlant_costs.csv` defines cost data for all technologies in each country. It includes:

- Capital expenditure (CAPEX) in USD/MW (currency based on input data)
- Fixed operation and maintenance cost (FOM) in USD/MW (currency based on input data)
- Variable operation and maintenance cost (VOM) in USD/MWh (currency based on input data)

Note: Currencies may vary depending on the source data.

This data applies to generators, storage, converters, and storage capacity expansion (In case of lithium battery, it refers to inverter costs).

Renewables Technical Potential

`Renewables_technical_potential.csv` defines maximum expansion limits (technical potential or land-use limits) for renewable technologies. It is currently only applied to solar photovoltaic (`PHOT`), hydro run-of-river (`HROR`), onshore wind (`WTON`), offshore wind (`WTOF`), rooftop PV (`RTPV`), solar hot water heater (`SWHT`) but can be modified to apply for other technologies.

The model builder does not allow higher expansion than what are specified in this CSV file. You can expand the file to include other technologies if needed.

Storage Costs

`Storage_costs.csv` covers the cost structure for all storage tanks (storage volume or energy capacity) in each country. It includes:

- Capital expenditure (CAPEX) in USD/MW (currency based on input data)
- Fixed operation and maintenance cost (FOM) in USD/MW (currency based on input data)
- Variable operation and maintenance cost (VOM) in USD/MWh (currency based on input data)

Note: Currencies may vary depending on the source data.

Storage Inflows

`Storage-inflows.csv` provides time-series inflow data [MW] for `StorageUnit` components. Inflow profiles are only designed for reservoir-based systems like hydropower and hydro pumped storage.

Matching the inflows is based on the technology (hydro dam (`HDAM`) and/or hydro pumped storage (`HPHS`)) and their location (e.g., region `XY_NO` in country `XY`). If the technology is listed and it requires inflow profile, but the profile is not in this csv, then it will be defined as constant 0 for all hours.

3.3.3 Input Data: Regional CSV Template

Structure of the regional CSV template files

```
data
└ example
  └ project_01
    └ scenario_01
      ├ Power
      | ├ buses.csv
      | ├ decommission_capacity.csv
      | ├ fuel_suppliers.csv
      | ├ interconnector.csv
      | ├ loads.csv
      | ├ power_generators.csv
      | ├ power_links.csv
      | ├ storage_capacity.csv
      | └ storage_energy.csv
      ├ Industry
      | ├ buses.csv
      | ├ decommission_capacity.csv
      | ├ direct_air_capture.csv
      | ├ fuel_conversion.csv
      | ├ heat_generators.csv
      | ├ heat_links.csv
      | ├ loads.csv
      | ├ storage_capacity.csv
      | └ storage_energy.csv
      └ Transport
        ├ buses.csv
        ├ loads.csv
        ├ pev_chargers.csv
        └ pev_storages.csv
```

> Tip
>
> The currency of all example data is `USD` defined in the `base_configs` section of `config.yaml`. You can refer to Model Builder Configuration for more information.

> Tip
>
> If there's a cell with `inf` in the csv files, it represents infinite value in `float` datatype when it is read into the network.

Buses

This file defines the buses to be used in the model. All components need to be connected to one or more buses.

Parameter	definition
`country`	2-letter country codes according to ISO 3166 format
`node`	Node or region name within the country (can be the same as country if the model is not region-specific)
`bus`	PyPSA `bus` component. Format: `{NODE}_{TECHNOLOGY}N` or `{COUNTRY}_{TECHNOLOGY}N` (suffix `N` is not applied for technologies with `HVELEC`, `LVELEC`, `IND-LH`, `IND-HH`, or `ATMP`)
`carrier`	Fuel or resource name. Unlike other entries that are in uppercase, here only the first letter is capitalized.

Decommission Capacity

`Decommission_capacity.csv` contains the installed capacity of power plants scheduled for decommissioning.

- For Generators, the `name` column must match the `name` column in `input_dir/project_name/scenario_name/Power/power_generators.csv`.
- For Links, the `name` column must match the `link` column in `input_dir/project_name/scenario_name/Power/power_links.csv`

Parameter	definition
`country`	2-letter country codes according to ISO 3166 format
`name`	Asset name (to be decommissioned)
`class`	Component type of the asset in PyPSA network. Only the first letter is capitalized
`years`	Decommission plan in each year [MW]

Fuel Supplies

These are fuel supply generators that provide fuel in the thermal energy unit [MWh_th]. It is possible to put maximum supply constraint over a year for these fuel supplies. See model schematic diagram

Parameter	definition
`country`	2-letter country codes according to ISO 3166 format
`bus`	Fuel supply `bus`. Format: `{COUNTRY}_{TECHNOLOGY}N` (suffix `N` is not applied for technologies with `HVELEC`, `LVELEC`, `IND-LH`, `IND-HH`, or `ATMP`)
`supply_plant`	Fuel supply hub. Format: `TGEN_{TECHNOLOGY}N`
`carrier`	Fuel or resource name
`max_supply`	Annual fuel supply limit [MWh/year]
`fuel_cost`	Fuel cost [CURRENCY/MWh]
`year`	Year of the fuel supply

Interconnector

Interconnectors connect different regions by their maximum power transfer capacity.

Parameter	definition
`country`	2-letter country codes according to ISO 3166 format
`link`	Name of the interconnection link between two regions/countries. Format: `{NODE}_HVELEC_to_{NODE}_HVELEC`
`bus0`	Region/country exporting electricity to `bus1`. Used as the `bus` component in the PyPSA network Format: `{NODE}_HVELEC`
`bus1`	Region/country importing electricity from `bus0`. Used as the `bus` component in the PyPSA network Format: `{NODE}_HVELEC`
`carrier`	Energy carrier or resource (e.g., electricity, gas). Only the first letter is capitalized
`type`	Interconnector technology (e.g., `ITCN`). All uppercase
`efficiency`	Efficiency of the interconnector
`p_max_pu`	Maximum availability per snapshot (per unit of `p_nom`)
`p_min_pu`	Minimum availability per snapshot (per unit of `p_nom`)
`p_nom`	Nominal capacity in the default year [MW]
`p_nom_extendable`	Indicates if capacity can be expanded. Possible values: `TRUE` or `FALSE`
`CAP`	Capital expenditure in USD/MW (currency based on input data)
`FOM`	Fixed annual operation and maintenance cost in USD/MWa (currency based on input data)
`marginal_cost`	Marginal cost of the link in USD/MWh (currency based on input data)
`p_nom_max_{YEAR}`	Maximum additional capacity allowed for the given year in MW
`p_nom_min_{YEAR}`	Minimum additional capacity allowed for the given year in MW

Loads

This file contains the total load per load type which is matched to `profile_type`. You can connect multiple load types to the same bus and they would be added to create total load for the bus.

Parameter	definition
`country`	2-letter country codes according to ISO 3166 format
`node`	Name of the nodes or regions
`bus`	PyPSA `bus` component. The values can be `{NODE}_HVELEC` or `{NODE}_LVELEC` for power sector, `{NODE}_IND-LH` or `{NODE}_IND-HH` for industry sector, and `{NODE}_TRAN-PRV` or `{NODE}_TRAN-PUB` for transport sector
`profile_type`	Load profile type
`name`	Load name. Format: `{BUS}_{PROFILE_TYPE}`
`total_load`	Total annual load in MW
`carrier`	Energy carrier or resource. First letter capitalized
`year`	Year of the load data

Generators

See details of implementation in power and industry sectors.

Parameter	definition
`country`	2-letter country codes according to ISO 3166 format
`node`	Name of the nodes or regions
`type`	Generator technology.
`carrier`	Energy carrier or resource. First letter capitalized.
`bus`	PyPSA `bus` component. Format: `{NODE}_{TECHNOLOGY}N` (suffix `N` is not applied for technologies with `HVELEC`, `LVELEC`, `IND-LH`, `IND-HH`, or `ATMP`)
`name`	Generator name. Format: `{BUS}_{TECHNOLOGY}`
`p_nom`	Nominal capacity in the default year in MW
`p_nom_extendable`	Indicates if capacity can be expanded. Possible values: `TRUE` or `FALSE`
`p_nom_max_{YEAR}`	Maximum additional capacity allowed in the given year in MW
`p_nom_min_{YEAR}`	Minimum additional capacity allowed in the given year in MW

Links

See details of implementation logic in power, industry, transport sectors.

Parameter	definition
`country`	2-letter country codes according to ISO 3166 format
`bus0...3`	PyPSA `bus` components. Format: `{NODE}_{TECHNOLOGY}N` (suffix `N` is not applied for technologies with `HVELEC`, `LVELEC`, `IND-LH`, `IND-HH`, or `ATMP`)
`type`	Link technology
`link`	Link name. Format: `{BUS0}_to_{BUS1}_by_{TECHNOLOGY}`
`carrier`	Energy carrier or resource. First letter capitalized
`p_nom`	Nominal capacity in the default year in MW
`p_nom_extendable`	Indicates if capacity can be expanded. Possible values: `TRUE` or `FALSE`
`p_nom_max_{YEAR}`	Maximum additional capacity allowed in the given year in MW
`p_nom_min_{YEAR}`	Minimum additional capacity allowed in the given year in MW

Storage Capacity

See details of implementation logic for power and industry sectors.

Parameter	definition
`country`	2-letter country codes according to ISO 3166 format
`node`	Name of the nodes or regions
`type`	Storage technology
`carrier`	Energy carrier or resource. First letter capitalized
`bus`	PyPSA `bus` component. The values can be `{NODE}_HVELEC` or `{NODE}_LVELEC` for power sector, and `{NODE}_IND-LH` for industry sector
`name`	Storage name. Format: `{BUS}_{TECHNOLOGY}`
`p_nom`	Nominal capacity in the default year in MW
`p_nom_extendable`	Indicates if capacity can be expanded. Possible values: `TRUE` or `FALSE`
`p_nom_max_{YEAR}`	Maximum additional capacity allowed in the given year in MW
`p_nom_min_{YEAR}`	Minimum additional capacity allowed in the given year in MW

Storage Energy

See details of description for use of storage energy in power sector and industry sectors.

Parameter	definition
`country`	2-letter country codes according to ISO 3166 format
`bus`	PyPSA `bus` component. Format: `{NODE}_{TECHNOLOGY}N` (suffix `N` is not applied for technologies with `HVELEC`, `LVELEC`, `IND-LH`, `IND-HH`, or `ATMP`)
`store`	Energy storage name. Format: `{BUS}_STOR`
`type`	Storage technology
`carrier`	Energy carrier or resource. First letter capitalized
`standing_loss`	Hourly energy loss rate during storage, in %/hour.
`e_nom`	Nominal energy in the default year in MWh
`e_nom_extendable`	Indicates if capacity can be expanded. Possible values: `TRUE` or `FALSE`
`max_store_{YEAR}`	Maximum additional capacity allowed in the given year in MW

EV Chargers

See details of implementation here.

Parameter	definition
`country`	2-letter country codes according to ISO 3166 format
`link`	Link name. Format: `{BUS0}_to_{BUS1}_by_{TECHNOLOGY}`
`bus0`	Region/country exporting electricity to `bus1`. Used as the `bus` component in the PyPSA network Format: `{NODE}_LVELEC`
`bus1`	Region/country importing electricity from `bus0`. Used as the `bus` component in the PyPSA network Format: `{NODE}_TRAN-PRV` or `{NODE}_TRAN-PUB`
`type`	Storage technology: private (`EVCH-PRV`) or public (`EVCH-PUB`)
`carrier`	Fixed as `Electricity`
`p_max_pu`	Profile reading from `availability.csv` based on private (`EVCH-PRV`) or public (`EVCH-PUB`)
`num_ch_{YEAR}`	Number of electric vehicles charged in the given year

EV Storages

See details of implementation here.

Parameter	definition
`country`	2-letter country codes according to ISO 3166 format
`node`	Name of the nodes, regions, or countries
`type`	Storage technology: private (`EVST-PRV`) or public (`EVST-PUB`)
`carrier`	Fixed as `Electricity`
`bus`	PyPSA `bus` component. Format: `{NODE}_TRAN-PRV` or `{NODE}_TRAN-PUB`
`name`	Storage name. Format: `{BUS}_{TYPE}`
`num_ev_{YEAR}`	Number of electric vehicles in the given year

3.3.4 Input Data: Model Builder Configuration

You can configure several settings for running PyPSA-SPICE in `config.yaml` file located in the root directory of the project. We suggest to make one config file per scenario you are running. This section provides a detailed explanation of the key variables that can be configured to run different scenarios.

Init Settings

Path configurations

```yaml
path_configs:
  input_dir: data/example/ #(1)!
  results_dir: results/ # (2)!
  project_name: project_01 #(3)!
  scenario_name: scenario_01 # (4)!
```

1. Directory containing all scenario data. Inside this folder, subfolders for `project_name` and `scenario_name` will be created.
2. Directory where output network files, CSVs and graphs are saved.
3. Directory for project-related data, including the `scenario_name` folder.
4. Directory for storing scenario input CSVs.

The path for the skeleton folder follows the pattern: `input_dir` / `project_name` / `scenario_name`.

This config file is used for both creating a new model via `python script/build_skeleton.py` (see the section on defining a new model) and used for running different instances of the model.

> **Tip**
>
> Make sure your snakemake file points to correct config file. To run different scenarios, you just need to change the snakemake file to the corresponding scenario config file.

Base configurations

```yaml
base_configs:
  countries: ["XY", "YZ"] # (1)!
  regions: # (2)!
    XY: ["NR","CE", "SO"]
    YZ: ["NR","CE", "SO"]
  years: [2025, 2030, 2035, 2040, 2045, 2050] # (3)!
  sector: ["p-i-t"] # (4)!
  currency: USD # (5)!
```

1. A list containing 2-letter country codes according to ISO 3166.
2. List of regions or nodes within each country. This defines the network's nodal structure.
3. List of years to be executed in the model builder.
4. List of sectors to include in model run. The power sector (`p`) needs to be included. Other available options are `p-i`, `p-t`, `p-i-t`, representing industry (`i`), and transport (`t`) sectors coupled with the power sector.
5. Currency usd in the model. The default setting is USD (also used in example data). Format shall be in all uppercases, ISO4217 format.

Scenario Settings

Scenario configurations

```yaml
scenario_configs:
  snapshots: # (1)!
    start: "2025-01-01"
    end: "2026-01-01" # (2)!
    inclusive: "left" # (3)!
  resolution:
    method: "nth_hour" # (4)!
    number_of_days: 3 # (5)!
    stepsize: 25 # (6)!
```

```
  interest: 0.05  # (7)!
  remove_threshold: 0.1 # (8)!
```

1. Defines the start and end dates for the model's time period. Dates are in "YYYY-MM-DD" format.
2. The model performs optimization on a yearly basis, with each modeling year defined as 8760 hours for hourly resolution. If the selected base year is a leap year, it is recommended to set the end date to `-12-31` of that year.
3. Defines which side of the selected snapshot should be included in the model builder. In the given example, if this parameter is set to `left`, the zeroth hour of the start time snapshot i.e. `2025-01-01 00:00` will be included in the model while the `2026-01-01 00:00` will not be included. We recommend to leave this as is.
4. Determines the method used by the model to deal with the time steps. For testing this reduces the compute time. Available options are `nth_hour` (recommended) and `clustered`. Depending on the selected option, one other parameter in the `resolution` section should be set.
5. If `method = "clustered"`, the number of representative days should be provided to group time steps and form the model builder timestamps. For example, if `number_of_days: 3`, the model builder will only solve 72 hours in the entire year.
6. This is used when `method = "nth_hour"`. In this case, the model builder will run at every n-th hour. Typical value to use for this would be 25, so every 25th hour is included in the model. To run the model at hourly resolution (the highest temporal resolution in the model builder), then it needs to be set to 1.
7. Interest rate in decimal form (e.g., 0.05 represents 5%).
8. Removes non expandable assets with a capacity below this threshold (in MW) to avoid numerical issues during optimization.

Mandatory Constraints

The CO_2 management is the most important and mandatory constraint in the model. The model allows for two different instruments to decrease CO_2 emissions: CO_2 price or CO_2 constraint. You can choose between each of these but not both. The variables listed below should be filled out for each country individually.

Constraints for CO_2 management

```
co2_management: # (1)!
  XY:
    option: "co2_cap" # (2)!
    value:
      2025: 100
      2030: 90
      2035: 130
      2040: 110
      2045: 100
      2050: 90
  YZ:
    option: "co2_price" # (3)!
    value:
      2025: 1
      2030: 1
      2035: 1
      2040: 1
      2045: 1
      2050: 1
```

1. Please indicate country/region specific CO_2 management mode: `"co2_cap"` or `"co2_price"`.
2. Goal is to minimize the CO_2 emissions in each year and target values are given in $mtCO_2$.
3. Goal is to minimize the emission price (`"co2_price"`) in the system and target values are in $\$/tCO_2$.

> Tip
>
> If you don't want to use any CO_2 constraint, default to using `co2_price` with very small value for the years.

Custom Constraints

The custom constraints section allows you to apply additional rules or limits to the model's behavior, tailoring it to specific scenario requirements. All custom constraints are listed below in the two countries as an example. These constraints can control various aspects of the model, such as renewable generation share, thermal power plant operation, reserve margins, energy independence, and production limitations.

By adjusting these settings, you can implement assumptions or policies. The settings listed below should be configured for each country individually.

> **Note**
>
> By default these are not included, so if you need a custom constraint, the corresponding part needs to be included in your scenario config file.

Custom constraints

```yaml
custom_constraints:
  XY:
    energy_independence: # (1)!
      pe_conv_fraction: # (2)!
        Solar: 1
        Wind: 1
        Geothermal: 1
        Water: 1
      ei_fraction: # (3)!
        2025: 0.3
        2030: 0.4
        2035: 0.5
        2040: 0.6
        2045: 0.7
        2050: 0.8
    production_constraint_fuels: ["Bio", "Bit", "Gas", "Oil"] # (4)!
    reserve_margin: # (5)!
      epsilon_load: 0.1 # (6)!
      epsilon_vre: 0.1 # (7)!
      contingency: 1000 # (8)!
      method: static # (9)!
    res_generation: # (10)!
      math_symbol: "<=" # (11)!
      res_generation_share: # (12)!
        2030: 0.25
        2035: 0.35
        2040: 0.4
        2045: 0.45
        2050: 0.5
    thermal_must_run: # (13)!
      must_run_frac: 0.2 # (14)!
  YZ:
    capacity_factor_constraint: # (15)!
      "SubC": 0.6
      "SupC": 0.6
      "HDAM": 0.4
    production_constraint_fuels: ["Bio", "Bit", "Gas", "Oil"]
    res_generation:
      math_symbol: "<="
      res_generation_share:
        2030: 0.1
        2035: 0.2
        2040: 0.3
```

```
        2045: 0.3
        2050: 0.3
```

1. The model adds constraints to ensure energy independence. This indicates how much of energy needs are met without relying on imports (by producing enough energy domestically). You can refer to Constraint - Energy Independence for more information.
   To deactivate it, you can exclude them in the `custom_constraints`, and the model will identify it as deactivated.
2. Primary energy conversion factor (dimensionless) is used to convert electricity generation to `primary energy` to make renewables comparable to fossil at primary energy level. Different definitions can be used to arrive at the value of these.
3. Minimum energy independence fraction defined as: $$ \frac{\textit{locally produced energy}}{\textit{locally produced energy + imported energy}} $$ For details see `Energy Independence constraint` below.
4. Maximum production limit of certain fuels can be defined here. Maximum values for these fuels are defined in `Power/fuel_supplies.csv`.
   To deactivate it, you can remove them in the `custom_constraints`, and the model will identify it as dectivated.
5. The model adds reserve margin constraints based on `reserve_parameters`. See Constraint - Reserve Margin for more information.
   To deactivate it, you can exclude them in the `custom_constraints`, and the model will identify it as dectivated.
6. Fraction of load considered as reserve.
7. Contribution of Variable Renewable Energy (VRE) to the reserve.
8. Extra contingency in MW. It is under `reserve_margin`. This is usually taken as a the size of largest individual power plants or defined by country specific regulations.
9. Options: `static` (no VRE) or `dynamic` (includes VRE). See reserve margin definition below.
10. The model adds a constraint on renewable generation as a fraction of total electricity demand.
11. Defines the type of renewable constraint. If set to `<=` it means the fraction of renewable generation from the total electricity demand should be less than or equal to the given values or greater than or equal to if value set to `=>`
12. Fraction of renewable generation to the total electricity demand for each year.
13. The model forces combined thermal power plants to have minimum generation level as a fraction of load.
14. Fraction of thermal generation to the total electricity demand per snapshot providing the baseload.
15. Maximum capacity factor of certain technologies can be defined here.
    To deactivate it, you can exclude them in the `custom_constraints`, and the model will identify it as dectivated.

In the following two sub-sections, we provide more information about the definition of energy independence and reserve margin.

ENERGY INDEPENDENCE: MATHEMATICAL FORMULATION

This constraint forces the model to keep the ratio of locally produced power to the sum of locally produced power and imported power to be more than the minimum energy independence factor:

$$\frac{loc}{imp + loc} \ge \phi \qquad \Longrightarrow \qquad (1 -\phi) \cdot loc >= \phi \cdot imp$$

Where

parameter	description	mathematical formulation
imp	Imported power: Generation from the theoretical import fuel-based generators	$\sum_{f \in F} \sum_{t \in T} {G_t^{TGEN-f-import}}$
loc	Local power generation: Fuel-based generations from local resources + renewable generations x primary energy conversion factor	$\sum_{f \in F} \sum_{t \in T} {G_t^{TGEN-f-local}} + \alpha_{res}*\sum_{t}G_{res}$
ϕ	minimum energy independence factor	
α_{res}	Primary energy conversion factor used for renewable sources for electricity generation. This value can be 0-1 for renewables, or larger than 1 for other generation sources depending on the energy policy in the country.	

RESERVE MARGIN: MATHEMATICAL FORMULATION

The reserve margin constraint in PyPSA-SPICE is modeled similarly to the GenX approach.

$$\sum_g r_{g,t} \ge \epsilon^{load} \cdot \sum_n d_{n,t} + \epsilon^{vres} \cdot \sum_{g \in \mathcal{G}^{VRES}} \bar g_{g,t} \cdot G_g + {contingency} \qquad \forall t$$

Where

parameter	description
$r_{g,t}$	Reserve margin of generator `g` at time `t`
ϵ^{load}	Fraction of load considered for reserve
$d_{n,t}$	Demand at node `n` and time `t`
ϵ^{vres}	Fraction of renewable energy for reserve
$\bar g_{g,t}$	Forecasted capacity factor for renewable energy of generator `g` at time `t`
G_g	Capacity of generator `g`
$\mathcal{G}^{VRES}$	Set of renewable generators in the system
$contingency$	Fixed contingency

See Linopy example of the reserve constraint implementation for more details.

Solver Settings

Solving the optimisation model builder requires a good solver to boost the performance. PyPSA-SPICE supports solvers such as `gurobi`, `cplex`, and `highs`. A comparison of solver performance is available in solver benchmarking results.

Solver configurations

```yaml
solving:
  solver:
    name: cbc #(1)!
    options: cbc-default #(2)!
  oetc: # (3)!
    activate: false
    name: test-agora-job
    authentication_server_url: http://34.34.8.15:5050
    orchestrator_server_url: http://34.34.8.15:5000
    cpu_cores: 4
    disk_space_gb: 20
    delete_worker_on_error: false
  solver_options: #(4)!
    default: {}
    cbc-default:
      threads: 8 #(5)!
      cuts: 0 #(6)!
      maxsol: 1 #(7)!
      ratio: 0.1 #(8)!
      presolve: 1 #(9)!
      time_limit: 3600 #(10)!
    gurobi-default:
      threads: 8 #(11)!
      method: 2 #(12)!
      crossover: 0 #(13)!
      BarConvTol: 1.e-5 #(14)!
      AggFill: 0 #(15)!
      PreDual: 0 #(16)!
      GURO_PAR_BARDENSETHRESH: 200 #(17)!
    gurobi-numeric-focus:
      name: gurobi
      NumericFocus: 3 #(18)!
      method: 2 #(19)!
      crossover: 0 #(20)!
      BarHomogeneous: 1 #(21)!
      BarConvTol: 1.e-5 #(22)!
      FeasibilityTol: 1.e-4 #(23)!
      OptimalityTol: 1.e-4 #(24)!
      ObjScale: -0.5 #(25)!
      threads: 8 #(26)!
      Seed: 123 #(27)!
    cplex-default:
      threads: 4 #(28)!
```

```
    lpmethod: 4 #(29)!
    solutiontype: 2 #(30)!
    barrier_convergetol: 1.e-5 #(31)!
    feasopt_tolerance: 1.e-6
  highs-default: #(32)!
    threads: 4 #(33)!
    solver: "ipm"
    run_crossover: "off"
    small_matrix_value: 1e-6 #(34)!
    large_matrix_value: 1e9 #(35)!
    primal_feasibility_tolerance: 1e-5 #(36)!
    dual_feasibility_tolerance: 1e-5 #(37)!
    ipm_optimality_tolerance: 1e-4 #(38)!
    parallel: "on"
    random_seed: 123 #(39)!
  highs-simplex:
    solver: "simplex"
    parallel: "on"
    primal_feasibility_tolerance: 1e-5 #(40)!
```

```
    dual_feasibility_tolerance: 1e-5 #(41)!
    random_seed: 123 #(42)!
```

1. Compatible solvers are `gurobi`, `cplex`, `cbc`, and `highs`.
2. Depending on the selected solver, specify one of the corresponding options: `cbc-default`, `gurobi-default`, `gurobi-numeric-focus`, `cplex-default`, `highs-default`, or `highs-simplex`.
3. `oetc` is a colud computing service provided by Open Energy Trainsition Organization. To access and activate the service. Please reach out to us for more details.
4. Solver options can be adjusted in the following list. If no value is provided in the `solver/option` section, the default `solver_options`, which is empty, will be considered.
5. Number of CPU threads to be used by the solver for parallel computation to speed up solving time.
6. Cutting planes are typically used to tighten the problem and improve performance (usually beneficial in MIP problems). By disabling them solution will be obtained faster but potentially at the cost of optimality.
7. Limits the solver to finding only one solution. The solver will stop once it finds a feasible solution (instead of finding all solutions).
8. Specifies that the solver should stop if it finds a solution within 10% of the best possible bound. This is useful for faster solutions when absolute optimality is not required.
9. Enabling `presolve` simplifies the problem before starting the optimization to make it faster and more stable.
10. Sets a time limit for the solver (here 3600 seconds = 1 hour). The solver will stop if it exceeds this limit.
11. Number of CPU threads to be used by the solver for parallel computation to speed up solving time.
12. Algorithm used to solve continuous models or the initial root relaxation of a MIP model. `-1` chooses the algorithm automatically and other options are explained in Gurobi documentation for more information.
13. Barrier crossover strategy. `-1` chooses strategy automatically and `0` disables crossover, which will speed up the solution process but might reduce solution quality. Other options are explained in Gurobi documentation.
14. The barrier solver terminates when the relative difference between the primal and dual objective values is less than the specified tolerance. This parameter is in `[0, 1]` range and the default value is `1e-8`.
15. A parameter that controls the amount of fill allowed during the aggregation phase of presolve. `AggFill` determines how aggressively Gurobi merges constraints during aggregation. Higher values can potentially lead to more simplification but may also introduce numerical instability. `-1` chooses aggregation fill automatically and `0` disables it.
16. Determines whether the solver should dualize the problem during the presolve phase. Depending on the structure of the model, solving the dual can reduce overall solution time. The default setting (`-1`) decides about it automatically. Setting `0` forbids presolve from forming the dual, while setting `1` forces it to take the dual.
17. Sets the threshold for determining when a column in the constraint matrix is considered dense during barrier optimization. When the constraint matrix is dense, it means its non-zero elements are more than the `GURO_PAR_BARDENSETHRESH` value.
18. With higher values, the model will spend more time checking the numerical accuracy of intermediate results.
19. Algorithm used to solve continuous models or the initial root relaxation of a MIP model. `-1` chooses the algorithm automatically and other options are explained in Gurobi documentation for more information.
20. Barrier crossover strategy. `-1` chooses strategy automatically and `0` disables crossover, which will speed up the solution process but might reduce solution quality. Other options are explained in Gurobi documentation.
21. Setting the parameter to `0` turns it off, and setting it to `1` forces it on. The homogeneous algorithm is useful for recognizing infeasibility or unboundedness and is a bit slower than the default algorithm.
22. The barrier solver terminates when the relative difference between the primal and dual objective values is less than the specified tolerance. This parameter is in `[0, 1]` range and the default value is `1e-8`.
23. All constraints must be satisfied to a tolerance of `FeasibilityTol`. This parameter is in `[1e-9, 1e-2]` range and the default value is `1e-6`.
24. This parameter defines how close the solution needs to be to the best possible answer before the solver stops. It is in `[1e-9, 1e-2]` range and the default value is `1e-6`.
25. Positive values: divides the objective by the specified value to avoid numerical issues that may result from very large or very small objective coefficients. Negative values: uses the maximum coefficient to the specified power as the scaling (so ObjScale=-0.5 would scale by the square root of the largest objective coefficient). Default: `0` which means the model decides on the scaling automatically.
26. Number of CPU threads to be used by the solver for parallel computation to speed up solving time.
27. Fixed `Seed` values must be set if you want to get the same results (i.e., reproducibility of the optimization process). The value does not matter.

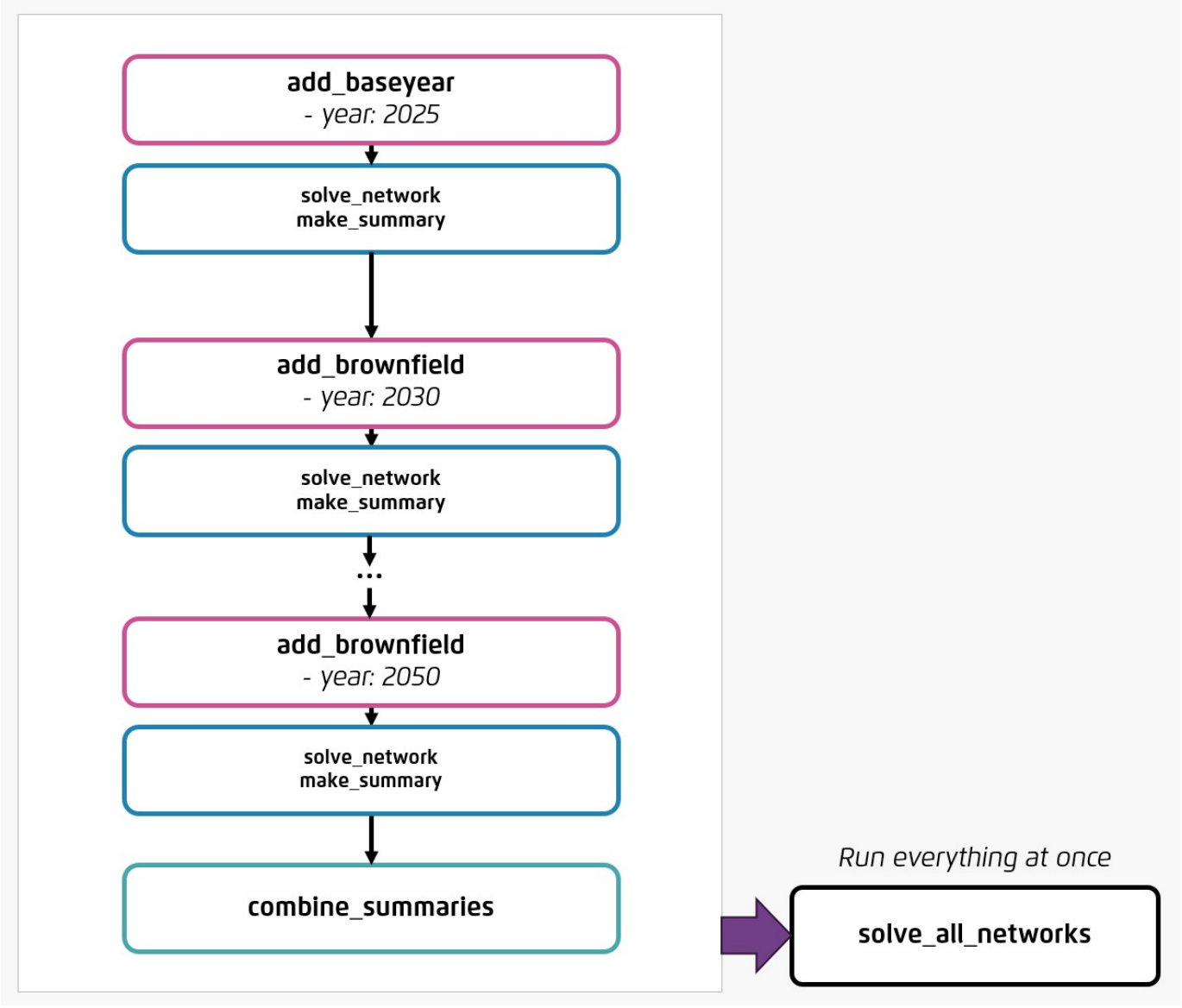
28. Number of CPU threads to be used by the solver for parallel computation to speed up solving time.
29. This parameter changes the algorithm and accepts an integer from `0` to `6`, where `0` denotes automatic choice of the algorithm, `1` is for primal simplex, `2` is for dual simplex, and `4` is for barrier.
30. Crossover can be turned off with `solutiontype=2` that instructs CPLEX not to seek a basic solution. This can be useful for a quick insight of the approx. optimal solution, if crossover takes long time.
31. Sets the tolerance on complementarity for convergence. Values can be qny positive number greater than or equal to `1e-12`; default: `1e-8`.
32. Please visit HiGHS documentation for complete list of options.
33. Number of CPU threads to be used by the solver for parallel computation to speed up solving time.
34. Values less than or equal to this will be treated as zero.
35. Values greater than or equal to this will be treated as infinite.
36. Range: `[1e-10, inf]`, default: `1e-07`.
37. Range: `[1e-10, inf]`, default: `1e-07`.
38. Range: `[1e-10, inf]`, default: `1e-07`.
39. Fixed `random_seed` values must be set if you want to get the same results (i.e., reproducibility of the optimization process). The value does not matter.
40. Range: `[1e-10, inf]`, default: `1e-07`.
41. Range: `[1e-12, inf]`, default: `1e-08`.
42. Fixed `random_seed` values must be set if you want to get the same results (i.e., reproducibility of the optimization process). The value does not matter.

3.3.5 Input Data: Model Execution

The execution of the model is orchestrated using Snakemake, as defined in the project's `Snakefile` located in the root directory. Snakemake acts as the workflow engine, coordinating individual rules and chaining them into a single, automated execution process. This allows for scalable, parallel computation across multiple cores or threads.

Execution of Snakemake workflow

Below is a simplified representation, showing the workflow for the years 2025-2050 with the power sector only.



Rules and Functions

The table below explains the main Snakemake rules and helper functions used in the PyPSA-SPICE workflow:

Name	Type	Description
`add_baseyear`	Snakemake rule	First step of building up the model after input data preparation is done. This rule imports all input data in the base year into the model. The base year is defined in `config.yaml` (see new-model or model-builder-configuration).
`add_brownfield`	Snakemake rule	Imports all output data from the previous year, using the `previous_year_outputs` and `solved_previous_year` functions. The definition of the different years is described in `config.yaml` (see new-model or model-builder-configuration).
`make_summary`	Snakemake rule	Generates output CSVs and summary of a specific year.
`combine_summaries`	Snakemake rule	Last step of the whole workflow aside from `solve_all_networks` rule. It generates summary of all years after all the networks from different years are solved.
`solve_network`	Snakemake rule	Solves the model builder for a specific year. This rule is only triggered after either `add_baseyear` or `add_brownfield`. Optimization settings are defined in `config.yaml` (see model-builder-configuration).
`solve_all_networks`	Snakemake rule	Executes the entire model builder workflow at once.
`previous_year_outputs`	Python Function	Reads the output CSVs of the previous year.
`solved_previous_year`	Python Function	Reads the network from the output of the previous year.

Running the Model Builder

RUN THE ENTIRE WORKFLOW AT ONCE

To execute the entire workflow, modify the `configfile` parameter in the snakemake to point to your scenario `config_scenario.yaml` file. After this, you can run the whole workflow using:

Run the entire model builder workflow at once.

```
snakemake -j1 -c4 solve_all_networks #(1)!
```

1. `-j1` means running only 1 job in parallel at a time and `-c4` means allowing each job to use up to 4 CPU threads. See the Snakemake CLI documentation to customize cores, threads, and parallel execution.

or

Running the entire model builder workflow using this call command

```
snakemake -call
```

RUN A SINGLE YEAR

If you want to run a specific year instead of running multiple years, You can do it using the following command:

Option 2: Run a single output file.

```
snakemake -j1 -c4 post-solve/elec_{SECTOR}_{YEAR}.nc
```

Since a particular year run needs previous year outputs (except base year), snakemake will run the all the required worksflows to generate the output for specified year.

4. 📊 Visualisation tool

4.1 PyPSA-SPICE-Vis: Visualisation tool for PyPSA-SPICE Model Builder

PyPSA-SPICE-Vis is an open-source tool designed to simplify the visualization and comparison of outputs generated by the PyPSA-SPICE model builder. It allows users to view and compare charts from post-analysis of multiple scenarios directly within a single webpage. As PyPSA-SPICE-Vis is built to work with the PyPSA-SPICE model builder, having the PyPSA-SPICE model builder is a prerequisite for using this tool.

4.1.1 How to use it

Create initial.yaml

The pypsa-spice-vis folder and its files should be inside the PyPSA-SPICE repository by default. The first step is to enter the pypsa-spice-vis folder and create an `initial.yaml` in the `setting` folder to provide initial settings for the app. This file will be ignored by git and is exclusive to your computer. You can refer to `pypsa-spice-vis/setting/initial_project_01.yaml` as an example, or copy the example file directly:

Copying the initial.yaml file

```
cp pypsa-spice-vis/setting/initial_project_01.yaml pypsa-spice-vis/setting/initial.yaml
```

Structure of the `initial.yaml` is explained as follows:

Explanation of initial.yaml

```
project: "project_01" #(1)!
sce1: "ref" # (2)!
sce2: "opt" #(3)!
```

1. Mandatory input project name. This name is the same as `project_name` from `pypsa-spice/config.yaml`.
2. Mandatory input name of the first scenario (the name is usually `ref` from the `scenario_name` in `pypsa-spice/config.yaml`).
3. Optional input name of the first scenario (the name is usually other scenarios from the `scenario_name` in `pypsa-spice/config.yaml`).

Run streamlit app

You can run streamlit app with the command below:

Current path in `pypsa-spice/` **Current path in `pypsa-spice/pypsa-spice-vis/`**

```
streamlit run pypsa-spice-vis/main.py

streamlit run main.py
```

After running, you will be able to see a local web link/url in the terminal. Simply open the link/url in your browser and you will be able to see the visualisations.

4.1.2 What charts are displayed in the tool by default

The available charts and sections are listed in List of charts and sections which you will be able to see all of them in the browser if your model include all sectors (power+industry+transport).

4.1.3 Deploy your visualistaion results to the web

You can review the visualisation in the localhost or deploy them into the webpage. More details are described in deployment

4.2 List of available sections and charts

The list of sections and available charts can be adjusted by `pypsa-spice-vis/setting/graph_settings.yaml` including maximum and minimum scales of the y-axis, units of the y-axis, etc.

4.2.1 Power

Chart name	Unit	Description
Capacity by type	GW	Installed capacity in the power sector by technology by modelling year
Capacity by region	GW	Installed capacity in the power sector by region by modelling year
Generation by type	TWh	Power Generation by technology by modelling year
Share category	%	Power Generation share by renewables & fossil fuels by modelling year
Transmission capacity between regions	GW	Capacity of transmission lines between different regions by modelling year
Hourly generation	MW	Hourly electricity generation by technology by modelling year
Regional hourly generation	MW	Hourly electricity generation by region by modelling year
Energy demand by carrier	TWh	Energy demand by carrier by modelling year
Hourly demand	MW	Hourly electricity demand by sector by modelling year
Hourly elec price	currency/MWh	Hourly marginal electricity price by region by modelling year
Hourly nodal flow between regions	MW	Hourly nodal exchange flow between regions by modelling year
Battery's E/P ratio	N/A	Hourly Energy-to-Power ratio of batteries by modelling year
Battery's charging profile	GW	Hourly charing/discharing status of batteries by modelling year

4.2.2 Industry

Chart name	Unit	Description
Capacity by carrier	GW	Installed capacity in the industry sector by technology by carrier by modelling year
Capacity by region	GW	Installed capacity in the industry sector by region by modelling year
Generation by region	TWh	Electricity generation used for industrial heat applications by region by modelling year
Generation by type	TWh	Electricity generation used for industrial heat applications by technology by modelling year

4.2.3 Transport

Chart name	Unit	Description
EV load profile	MW	Hourly charging load of electric vehicles in the tranport sector by modelling year

4.2.4 Emissions

Chart name	Unit	Description
Emission (power sector)	$MtCO_2$	Emission in the power sector by carrier by modelling year
Emission (industry sector)	$MtCO_2$	Emission in the industry sector by carrier by modelling year

4.2.5 Costs

Chart name	Unit	Description
CAPEX by type (power sector)	million currency	Capital Expenditure (CAPEX) in the power sector by technology by modelling year
Overnight investment by type (power sector)	million currency	Overnight investment in the power sector by technology by modelling year
CAPEX by type (industry sector)	million currency	Capital Expenditure (CAPEX) in the industry sector by technology by modelling year
CAPEX by type (transport sector)	million currency	Capital Expenditure (CAPEX) in the transport sector by technology by modelling year
CAPEX by type (all energy sectors)	million currency	Capital Expenditure (CAPEX) in all sectors by technology by modelling year
CAPEX and OPEX (all energy sectors)	million currency	Capital Expenditure (CAPEX) and Operational Expenditure in all sectors by modelling year
Average fuel costs (all energy sectors)	million currency/ $MWh_{thermal}$	Average fuel costs (thermal units) in all sectors by modelling year

4.2.6 Info

This section provides a list of all abbreviations and corresponding full names of all technologies and carriers that are used in the visualisation tool.

4.3 Deployment of PyPSA-SPICE-Vis app

The easiest method to deploy is using git-lfs to manage the data and simply using streamlit's community cloud.

1. Download git-lfs:

on Windows **on Linux** **On Mac**

```
download git-lfs directly from https://git-lfs.com/.

sudo apt-get install git-lfs

brew install git-lfs
```

1. To deploy the app, create a branch of the repo. This will also allow you to make changes like default values for country and scenario, graph setting, etc.
2. Install git-lfs via the following command:

installing git-lfs

```
git lfs install
```

1. Add the results data files inside this branch. These data files will be tracked using git-lfs. Note: add only the CSV files but do keep the same folder structure as that of `pypsa-spice` results.
2. Add path to this results folder inside `pypsa-spice-vis/setting/initial_project_01_deploy.yaml`. Note: the `pypsa-spice-vis/setting/initial.yaml` is ignored by git and used for local deployment.
3. In `pypsa-spice-vis/main.py`, make `DEPLOY == True`. By default, `False` for local run.
4. Initialise and track the result files using git-lfs. You will have to add all csvs to be tracked with lfs tracking system with following commands based on your operation system:

on Mac or Linux **on Windows**

```
git lfs install
git lfs track "*.csv"
git add .gitattributes
git commit -m "Track all CSV files with Git LFS"
find . -name "*.csv" -exec git add {} \;

git lfs install
git lfs track "*.csv"
git add .gitattributes
git commit -m "Track all CSV files with Git LFS"
for /r %i in (*.csv) do git add "%i"
```

After adding all csv files, you can initialise and commit tracked files using git-lfs:

committing all tracked files

```
git commit -m "Add all CSV files"
git push
```

Now the repository can be linked simply to Streamlit's deployment system.

5. 📚 Tutorials and Examples

5.1 Data Sources used in PyPSA-SPICE

There some several resources which can help you collect the input data:

- PyPSA technology-data
- The Danish Energy Agency: Technology Data for Generation of Electricity and District Heating

5.2 Tutorial Resources

PyPSA has a vibrant and welcoming community of modellers. Here are some resources which can help you understand the methodology, knowledge and structure of PyPSA related models:

- PyPSA introduction from TU Berlin
- PyPSA documentation
- PyPSA-meets-Earth discord channel
- PyPSA-Eur documentation

> Add your resource here?
>
> If you have a knowledge resource you would like to add, please share it with us by emailing at or add it to discussion forum on the PyPSA-SPICE repository.

5.2.1 PyPSA-SPICE or PyPSA Training Materials from Agora

We also provide training and long-term capacity building program to get you started. We have partner programs in various countries where we support using and building your PyPSA model. Therefore, feel free to reach out to us as we might be able to support you in getting started with PyPSA.

Some of the training material we use during our one off multi-day training and long-term capacity building program:

- Basic PyPSA training using green hydrogen production as an example.
- Training on PyPSA-SPICE

5.3 Publication using PyPSA-SPICE Model Builder

Here is a list of studies that have utilized the PyPSA-SPICE model. These studies were conducted either internally by Agora or in collaboration with our partners.

Publication	Link	Citation
Towards a collective vision of Thai energy transition: National long-term scenarios and socioeconomic implications	study link	
Alignment between Vietnam PDP8 and JETP commitment: Power system analysis 2030	study link	
Thailand: Towards carbon neutrality through PDP 2024: A cost-optimization analysis perspective	study link	
Kazakhstan's power system 2035: options for development	study link	

6. 🤝 Contributing

6.1 To be updated

6.2 To be updated

7. 🔗 References

7.1 Citing PyPSA-SPICE

Please use the citation below:

```
Agora Think Tanks (2025): PyPSA-SPICE: PyPSA-based Sector-Coupled Optimisation for Pathways Exploration
```

7.1.1 License

7.1.2 Contributions

We welcome anyone interested in contributing to this project, and please have a look at Contributing Guide and our Code of Conduct. If you have any ideas, suggestions or encounter problems, feel free to file issues or make pull requests on GitHub.

7.2 Developers & Reviewers

The following people have contributed significantly towards the development of PyPSA-SPICE:

7.2.1 Developers

Agora Energiewende

- Hai Long Nguyen
- Dr. Samarth Kumar
- Yu-Chi Chang
- Dr. Saeed Sayadi
- Dr. Jia Loy
- Thomas Kouroughli (former colleague at Agora Energiewende)

Technische Universität Berlin (TU Berlin)

- Dr. Elisabeth Zeyen

Institute for Climate and Sustainable Cities (ICSC)

- Jephraim Manansala

7.2.2 Reviewers

Technische Universität Berlin (TU Berlin)

- Dr. Fabian Neumann