

PyPSA-SPICE Model Builder

Documentation

Agora Think Tanks

Copyright © PyPSA-SPICE Developers

Table of contents

1. PyPSA-SPICE: PyPSA-based Scenario Planning and Integrated Capacity Expansion	5
1.1 Key Features of PyPSA-SPICE	5
1.2 Types of models outside the scope of PyPSA-SPICE	6
1.3 Studies conducted using PyPSA-SPICE	6
1.4 Visualisation of PyPSA-SPICE Results	6
1.5 Citing PyPSA-SPICE	6
1.6 Contributing	6
1.7 Maintained by	6
1.8 Supported by	6
1.9 License	7
2. User guide	8
2.1 Model builder methodology	8
2.2 Power sector	9
2.2.1 Key features	9
2.2.2 Power generators	10
2.2.3 Power links	10
2.2.4 Storage capacity	10
2.2.5 Storage energy	11
2.2.6 Carriers	12
2.2.7 Buses	13
2.2.8 Other components	13
2.2.9 Custom constraints (defined in the config.yaml file)	13
2.3 Industry sector	14
2.3.1 Key features	14
2.3.2 Heat generators	14
2.3.3 Heat links	15
2.3.4 Fuel conversion	15
2.3.5 Direct air capture	15
2.3.6 Storage capacity	15
2.3.7 Storage energy	15
2.3.8 Carriers	16
2.3.9 Buses	16
2.3.10 Other components	17
2.3.11 Custom constraints (defined in the config.yaml file)	17

2.4 Transport sector	18
2.4.1 Key features	18
2.4.2 Electric vehicle chargers	18
2.4.3 Electric vehicle storage	18
2.4.4 Carriers	19
2.4.5 Buses	19
2.4.6 Other components	19
2.4.7 Custom constraints (defined in the config.yaml file)	19
2.5 Standard output from the Snakemake workflow	20
2.5.1 NetCDF files (*.nc files)	20
2.5.2 Excel-ready CSVs	21
2.5.3 Jupyter Notebook	24
2.5.4 Dynamic visualisation	25
2.6 Troubleshooting	26
2.6.1 Setup debugger	26
2.6.2 Infeasibility issues	26
2.6.3 Tips for diagnosing the problem	26
2.6.4 More detailed information	26
3. Getting started	27
3.1 Overview of workflow	27
3.2 Installation	28
3.2.1 Clone the repository	28
3.2.2 Install Python dependencies	28
3.2.3 Install a solver	29
3.2.4 Set up the default configuration	29
3.2.5 Quick execution of the model builder using template data	29
3.3 Input data	31
3.3.1 Input data: define a new model	31
3.3.2 Input data: global CSV template	33
3.3.3 Input data: regional CSV template	36
3.3.4 Input data: model builder configuration	42
3.3.5 Input data: model execution	51
4. Visualisation tool	53
4.1 PyPSA-SPICE-Vis: visualisation tool for PyPSA-SPICE model builder	53
4.1.1 How to use it	53
4.1.2 What charts are displayed in the tool by default	53
4.1.3 Deploy your visualisation results to the web	53

4.2 List of available sections and charts	54
4.2.1 Power	54
4.2.2 Industry	54
4.2.3 Transport	54
4.2.4 Emissions	55
4.2.5 Costs	55
4.2.6 Info	55
4.3 Deployment of PyPSA-SPICE-Vis app	56
5. Tutorials and examples	57
5.1 Tutorial resources	57
5.1.1 PyPSA-SPICE or PyPSA training materials from Agora	57
5.2 Data sources used in PyPSA-SPICE	58
5.3 Publication using PyPSA-SPICE model builder	59
6. Contributing	60
6.1 How to contribute	60
6.1.1 Code style	60
6.1.2 Pre-commit	60
6.2 Code of conduct	61
6.2.1 Our pledge	61
6.2.2 Our standards	61
6.2.3 Enforcement responsibilities	61
6.2.4 Scope	61
6.2.5 Enforcement	61
6.2.6 Enforcement guidelines	62
6.2.7 Attribution	62
7. References	63
7.1 Citing PyPSA-SPICE	63
7.1.1 License	63
7.1.2 Contributions	63
7.2 Developers and reviewers	64
7.2.1 Developers	64
7.2.2 Reviewers	64

1. PyPSA-SPICE: PyPSA-based Scenario Planning and Integrated Capacity Expansion

license **GPL (>= 2)** | pypsa **v1.0.3** | snakemake **minimal==8.10.8** | code style **black**

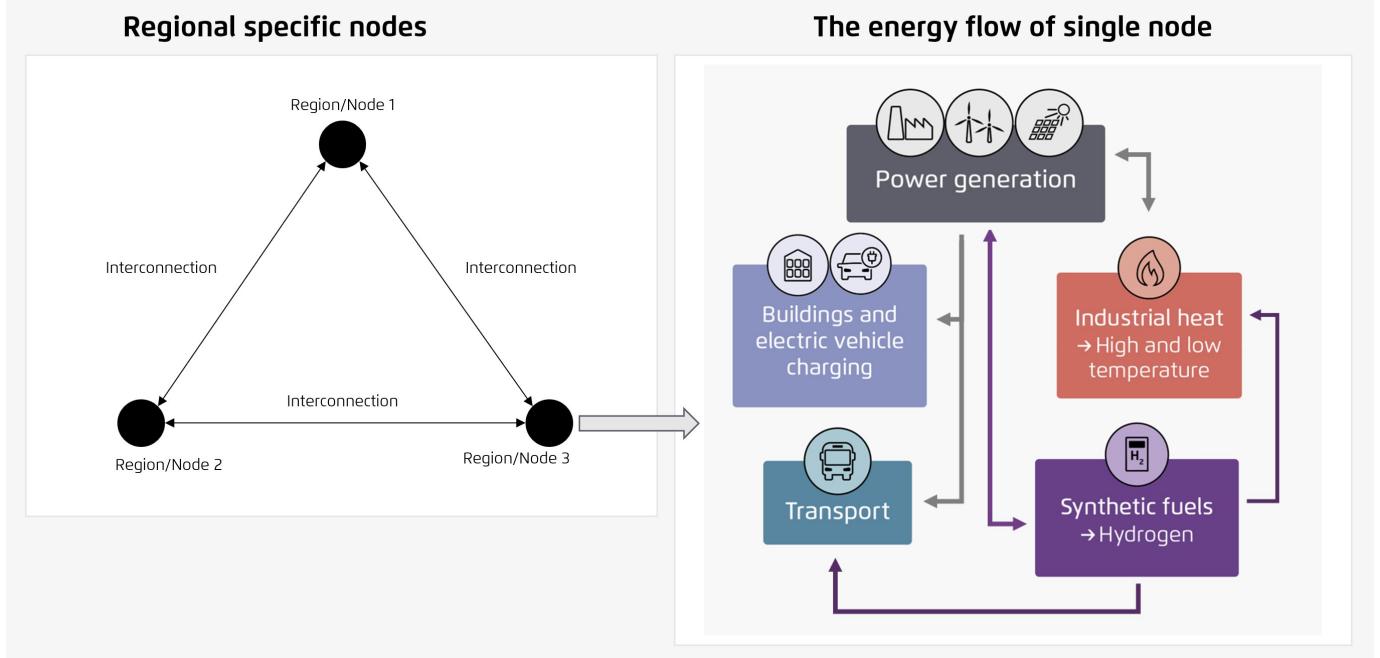
i Info

If you are considering using this model builder, please reach out to us at modelling@agora.thinktanks.org. 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 optimisation 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. Within each region, it models the integration of the power, heat, and transport sectors.

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

Example of an energy system model including 3 regions/nodes



1.1 Key Features of PyPSA-SPICE

- Assessment of national or regional mid-/long-term energy scenarios using a least-cost optimisation approach based on the **PyPSA** framework.
- Co-optimisation 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 visualisation 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 modelling power or energy systems with very high geographic resolution, such as load flow modelling. 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 Scenario Planning and Integrated Capacity Expansion

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 Maintained by



1.8 Supported by



1.9 License

Copyright © PyPSA-SPICE developers

PyPSA-SPICE is licensed under the open source [GNU General Public License v2.0 or later](#) with the following information:

The documentation is licensed under [CC-BY-4.0](#).

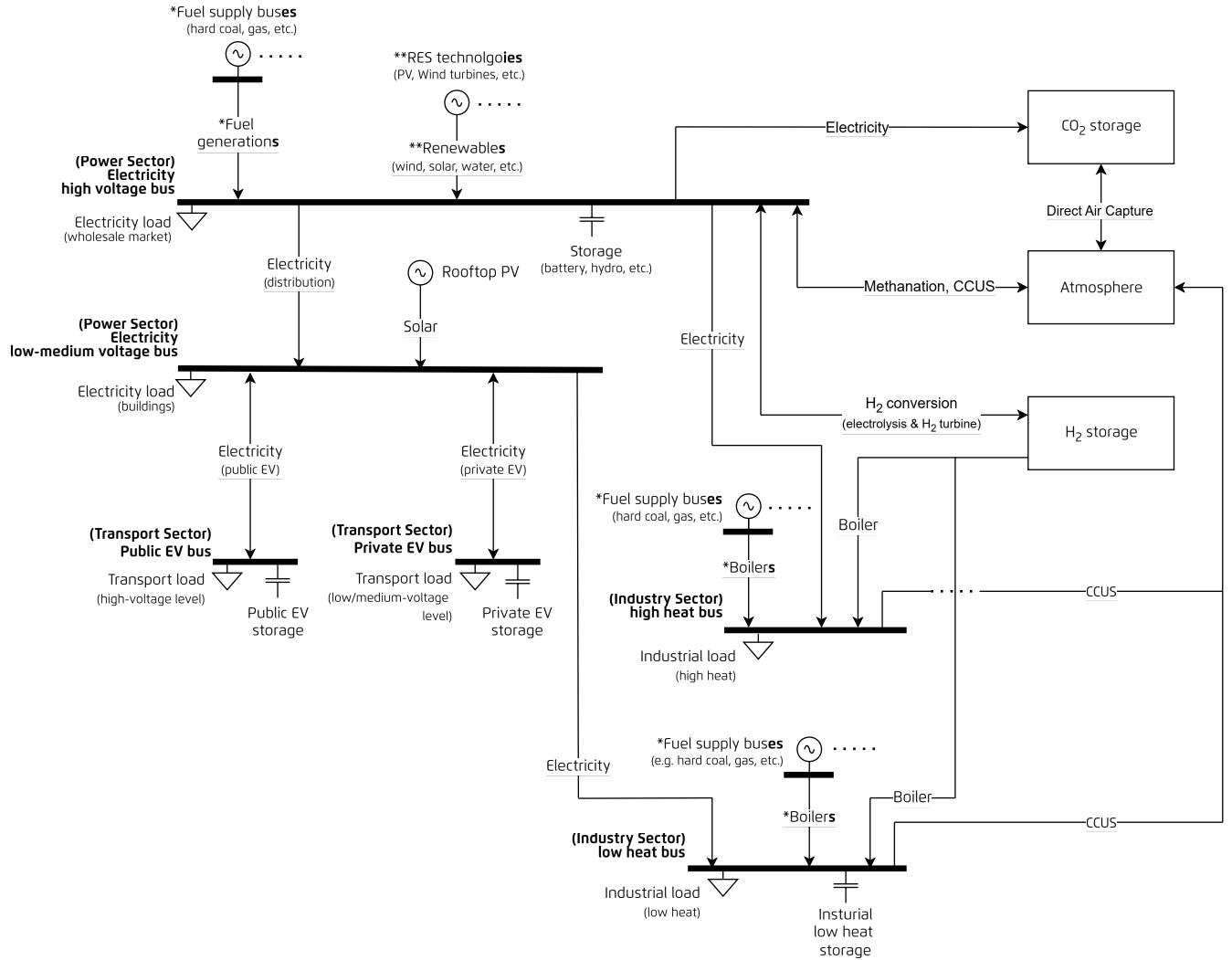
The repository uses [REUSE](#) to expose the licenses of its files.

2. User guide

2.1 Model builder methodology

PyPSA-SPICE is a least-cost optimisation 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 optimisation 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](#)

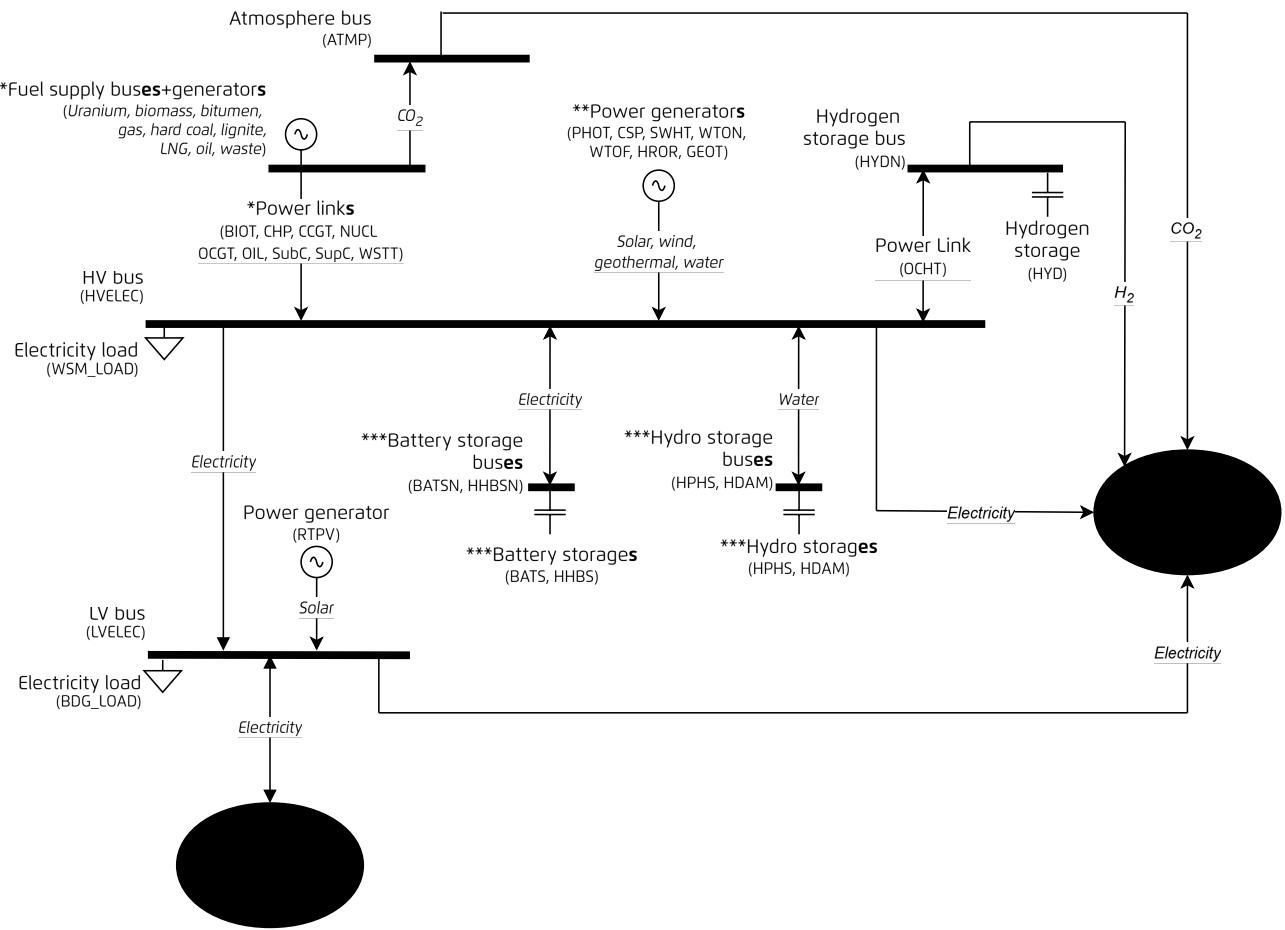
You can find more detailed information within each sector's dedicated section.

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-to-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-to-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 - StorageUnit](#) for more information). Thus, in PyPSA-SPICE model builder, hydro dam `HDAM` is defined as a `StorageUnit` and it is given in storage capacity 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
PHPSN	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.

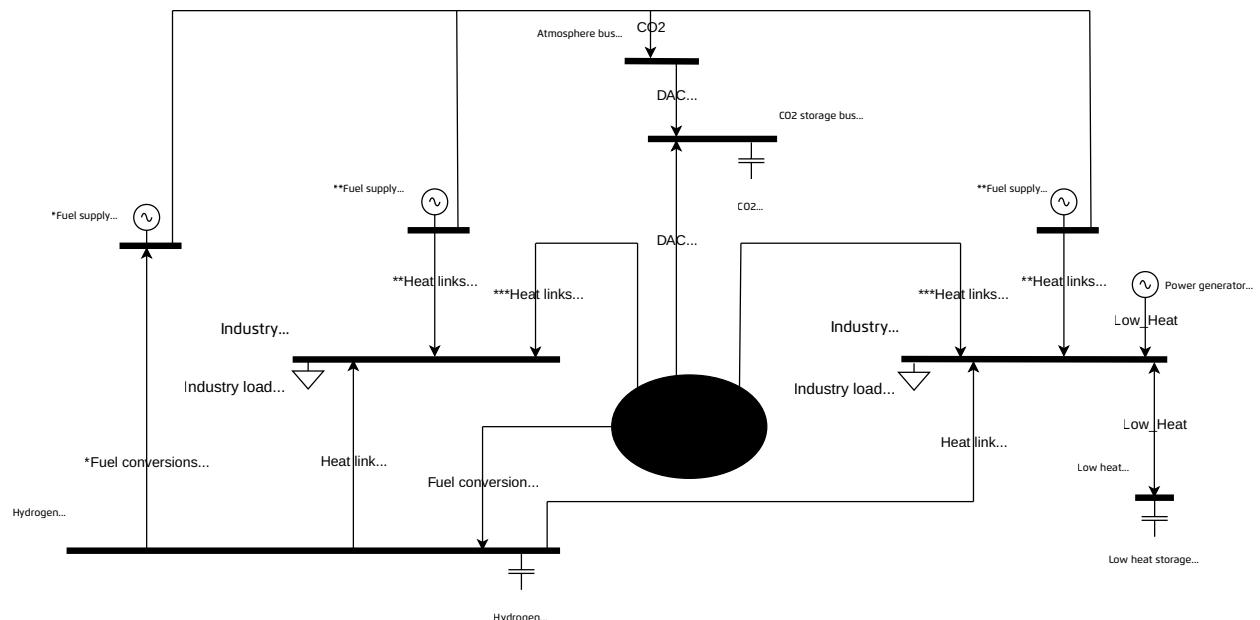
2.3 Industry sector

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...

Text is not SVG - cannot display

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, <code>StorageUnit</code> is modelled as a storage asset with a fixed energy-to-power ratio defined by <code>max_hours</code> of the nominal power (you can also refer to PyPSA Components - StorageUnit for more information).
To model the storage energy separately from the power capacity, <code>Store + 2 Links</code> 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.

In 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
<code>INLHSTOR</code>	Low-temperature heat storage

2.3.8 Carriers

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

2.3.9 Buses

Abbreviation	Full Name
<code>CO2STORN</code>	CO_2 storage
<code>HVELEC</code>	High-voltage electricity
<code>HYDN</code>	Hydrogen
<code>LVELEC</code>	Low-voltage electricity
<code>IND_LH</code>	Industrial low-temperature heat
<code>IND_HH</code>	Industrial high-temperature heat
<code>INLHSTORN</code>	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)

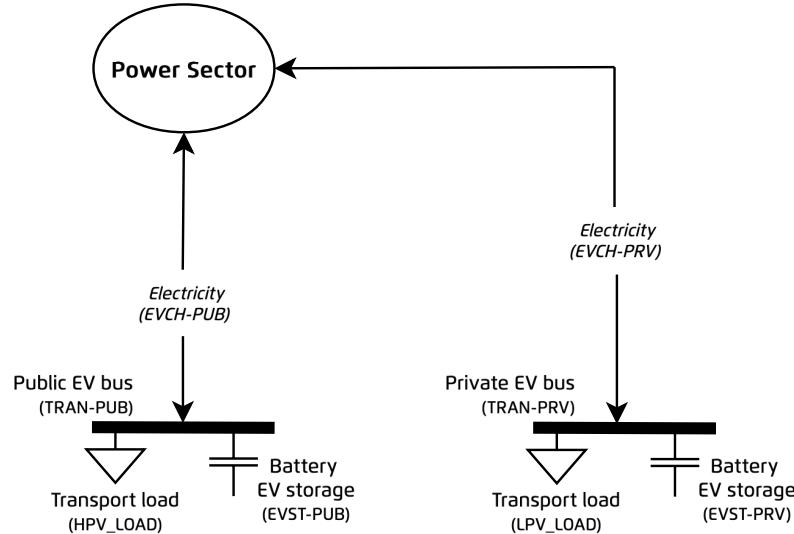
- coming soon...

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

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 modelling 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 modelling year
ene_costs_opex_capex_yearly	Energy	million currency	Total CAPEX and OPEX in energy sectors by modelling year
ene_dmd_by_carrier_yearly	Energy	TWh	Energy demand by carrier (e.g., Electricity) by modelling year
ene_emi_by_carrier_by_sector_yearly	Energy	MtCO ₂	Total emissions by carrier and sector by modelling year
ene_fom_by_type_yearly	Energy	million currency	Total fixed operation and maintenance cost by technology (e.g., CCGT) by modelling year
ene_gen_by_carrier_yearly	Energy	TWh	Total generation by carrier (e.g., Electricity) by modelling year
ene_opex_by_type_yearly	Energy	million currency	OPEX in energy sectors by technology (e.g., CCGT) by modelling year
ind_cap_by_carrier_by_region_yearly	Industry	GW	Installed capacity in the industry sector by carrier and region/node, and by modelling year
ind_cap_by_type_by_carrier_yearly	Industry	GW	Installed capacity in the industry sector by technology and carrier, and by modelling year
ind_emi_by_carrier_yearly	Industry	MtCO ₂	Emissions in the industry sector by carrier by modelling year
ind_gen_by_carrier_by_region_yearly	Industry	TWh	Generation in the industry sector by carrier and region/node, and by modelling 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 modelling year
ind_gen_by_type_hourly	Industry	MW	Generation in the industry sector by technology by modelling year
ind_hh_gen_by_type_hourly	Industry	MW	Generation in the industry sector (high-heat) by technology by modelling year
ind_lh_gen_by_type_hourly	Industry	MW	Generation in the industry sector (low-heat) by technology by modelling year
pow_bats_charging_hourly	Power	MW	Hourly battery charging profile by modelling 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 modelling year
pow_cap_by_region_yearly	Power	GW	Installed capacity in the power sector by region/node by modelling year
pow_cap_by_type_yearly	Power	GW	Installed capacity in the power sector by technology (e.g., CCGT) by modelling 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 modelling year
pow_capex_by_type_yearly	Power	million currency	CAPEX in the power sector by technology (e.g., CCGT) by modelling 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 modelling year
pow_flh_by_type_yearly	Power	Hours	Full load hours by technology (e.g., CCGT) by modelling year
pow_fom_by_type_yearly	Power	million currency	Fixed operation and maintenance cost in the power sector by technology (e.g., CCGT) by modelling year
pow_gen_by_category_share_yearly	Power	%	Fossil-Renewables share by modelling 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 modelling year
pow_gen_by_type_yearly	Power	TWh	Generation in the power sector by technology (e.g., CCGT) by modelling year
pow_hdam_flows_by_region_hourly	Power	MW	Hourly hydro dam flow profile by modelling year
pow_hpss_flows_by_region_hourly	Power	MW	Hourly hydro pumped storage profile by modelling year
pow_inter_cf_by_region_yearly	Power	-	Capacity factor of the interconnectors by region/node by modelling year
pow_intercap_by_region_yearly	Power	GW	

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

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., $\text{max} \geq \text{min}$).
- 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`

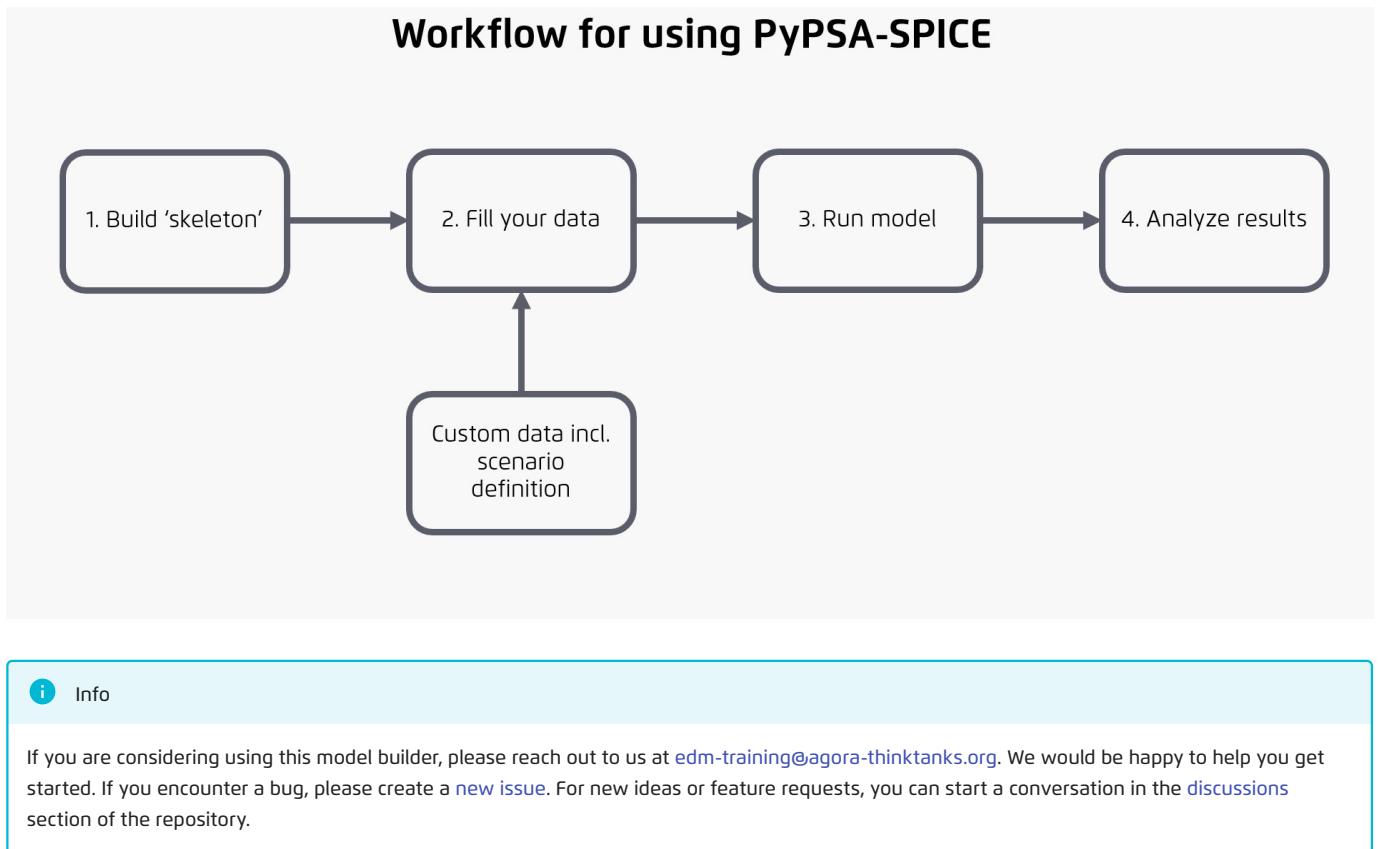
2.6.4 More detailed information

[PyPSA](#) also provides official guidance on solving infeasibility 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 visualisation. The following sections provide additional details and guidance for each step.



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

The network model created in PyPSA-SPICE model builder is passed to an external solver to perform total annual system cost minimisation 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 and open-source	Ipopt
Cbc	Free and open-source	Cbc
GLPK	Free and open-source	GLPK /WinGLPK
HiGHS	Free and 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 two configuration files:

1. `base_config.yaml`: Located in the project root directory, this file contains general configurations needed to set up the initial input data structure.
2. `scenario_config.yaml`: Located inside each scenario folder, this file contains scenario-specific configurations. It is only used after the input data structure has been created.

To get started, configure `base_config.yaml` first, then run the data setup process. Once complete, you can configure individual scenarios using their respective `scenario_config.yaml` files. For detailed configuration options, refer to [model-builder-configuration](#).

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 `data/pypsa-spice-data` folder can be used. You can 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. Adjust the information inside `base_config.yaml`.
2. Run `snakemake -c1 build_skeleton` to create a folder structure and template CSV files for your input data. All input folders and files shall be created inside the `data` folder after this command is executed.
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 base configuration file

Setting up the base 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 `base_config.yaml` contains two parts which will be used both for folder structure building and model executions:

Init settings in the config.yaml file

```
path_configs: #(1)
  data_folder_name: pypsa-spice-data
  project_name: project_01
  input_scenario_name: scenario_01 # (2)!
  output_scenario_name: scenario_01_tag1 # (3)!

base_configs:
  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 input scenario folder in your model.
3. A custom name you define for the output scenario folder to save your model results.
4. List of regions or nodes within each country. This defines the network's nodal structure. The country list contains 2-letter country codes according to [ISO 3166](#).
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:: `data / data_folder_name / project_name`.

By setting different `input_scenario_name` and country or regional settings in the `base_configs` section (see details in [Model configuration](#)), a new skeleton structure under the same `data_folder_name` 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

```
snakemake -c1 build_skeleton
```

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
└ pypsa-spice-data
  └ project_01
    └ input
      └ global_input
        ├ availability.csv
        ├ demand_profile.csv
        ├ ev_parameters.csv
        ├ power_plant_costs.csv
        ├ renewables_technical_potential.csv
        ├ storage_costs.csv
        ├ storage_inflows.csv
        └ technologies.csv
    └ 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
    scenario_config.yaml
results (will be created when during the model execution)

```

Tip

Once you've created a skeleton data folder for one scenario, you can simply duplicate the scenario folder and rename the folder name to set up additional scenarios. However, we recommend doing this only after you've completed filling in the data for the first one.

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 optimisations.

To help you fill these files:

- check [Global CSV template](#) for default file descriptions of country-level data.
- see [Regional CSV template](#) for detailed file descriptions of region-level data.

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 default, 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 across scenarios. This is to maintain comparability of the scenarios. `global_input_template` folder is used for the purpose of creating skeletons, and thus it can be considered as hidden folder inside the template.

Structure of the global CSV template files

```

data
└── global_input_template
    └── pypsa-spice-data
        └── project_01
            └── input
                └── global_input
                    ├── availability.csv
                    ├── demand_profile.csv
                    ├── ev_parameters.csv
                    ├── power_plant_costs.csv
                    ├── renewables_technical_potential.csv
                    ├── storage_costs.csv
                    └── storage_inflows.csv
            └── technologies.csv

```



Tip

The currency of all example data is `USD` defined in the `base_configs` section of `base_config.yaml`. You can refer to [Model builder configuration](#) for more information.

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 (`WT0N`), 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 normalised 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.

EV parameters

`ev_parameters.csv` stores the technical parameters relevant to the electric vehicles.

Power plant costs

`power_plant_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 the 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.

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
technology_nomenclature	Full names of the technology
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)

3.3.3 Input data: regional CSV template

Structure of the regional CSV template files

```

data
└ pypsa-spice-data
  └ project_01
    └ input
      └ 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

```

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

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 `data_folder_name/project_name/input/input_scenario_name/power/power_generators.csv`.
- For Links, the `name` column must match the `link` column in `data_folder_name/project_name/input/input_scenario_name/power/power_links.csv`.

Parameter	definition
<code>country</code>	2-letter country codes according to ISO 3166 format
<code>name</code>	Asset name (to be decommissioned)
<code>class</code>	Component type of the asset in PyPSA network. Only the first letter is capitalised
<code>years</code>	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
<code>country</code>	2-letter country codes according to ISO 3166 format
<code>bus</code>	Fuel supply bus. Format: {COUNTRY}_{TECHNOLOGY}N (suffix N is not applied for technologies with HVELEC, LVELEC, IND-LH, IND-HH, or ATMP)
<code>supply_plant</code>	Fuel supply hub. Format: TGEN_{TECHNOLOGY}N
<code>carrier</code>	Fuel or resource name
<code>max_supply</code>	Annual fuel supply limit [MWh/year]
<code>fuel_cost</code>	Fuel cost [CURRENCY/MWh]
<code>year</code>	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 capitalised
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_usd_mw	Capital expenditure in USD/MW (currency based on input data)
fom_usd_mwa	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_mwh	Total annual load in MWh
carrier	Energy carrier or resource. First letter capitalised
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 capitalised.
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 capitalised
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 capitalised
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 sector](#).

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 capitalised
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 in the [Transport sector](#).

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 in the [Transport sector](#).

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

PyPSA-SPICE requires two configuration files:

1. `base_config.yaml`: Located in the project root directory, this file contains general configurations needed to set up the initial input data structure.
2. `scenario_config.yaml`: Located inside each scenario folder, this file contains scenario-specific configurations. It is only used after the input data structure has been created.

To get started, configure `base_config.yaml` first, then run the data setup process. Once complete, you can configure individual scenarios using their respective `scenario_config.yaml` files.

base_config.yaml

Path configurations

```
path_configs:
  data_folder_name: pypsa-spice-data #(1)!
  project_name: project_01 #(2)!
  input_scenario_name: scenario_01 # (3)!
  output_scenario_name: scenario_01_tag1 # (4)!
```

1. Directory containing all scenario data. Inside this folder, subfolders for `project_name` and `input_scenario_name` will be created.
2. Directory for project-related data, including the `input_scenario_name` folder.
3. Directory for storing scenario input CSVs.
4. Directory for storing scenario output CSVs.

The path for the skeleton folder follows the pattern: `data_folder_name / project_name /input/ input_scenario_name`. The path for the output folder follows the pattern: `data_folder_name / project_name /results/ output_scenario_name`.

This config file is used for both creating a new model via `snakemake -c1 build_skeleton` (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

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

1. List of regions or nodes within each country. This defines the network's nodal structure. The country list contains 2-letter country codes according to [ISO 3166](#).
2. List of years to be executed in the model builder.
3. 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.
4. Currency usd in the model. The default setting is USD (also used in example data). Format shall be in all uppercases, [ISO4217](#) format.

scenario_config.yaml - scenario settings

Scenario configurations

```
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 optimisation on a yearly basis, with each modelling year defined as 8,760 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 optimisation.

scenario_config.yaml - mandatory constraints

The CO₂ management is the most important and mandatory constraint in the model. The model allows for two different instruments to decrease CO₂ emissions: CO₂ price or CO₂ 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 CO₂ 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 CO₂ management mode: `"co2_cap"` or `"co2_price"`.
2. Goal is to minimise the CO₂ emissions in each year and target values are given in mtCO₂.
3. Goal is to minimise the emission price (`"co2_price"`) in the system and target values are in USD/tCO₂.

 Tip

If you don't want to use any CO₂ constraint, default to using `co2_price` with very small value for the years.

scenario_config.yaml - 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

```
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: $\frac{\text{locally produced energy}}{\text{locally produced energy} + \text{imported energy}}$ 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 deactivated.
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 deactivated.
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 deactivated.

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:

$$\frac{loc}{imp + loc} \geq \phi \quad \Rightarrow \quad (1 - \phi) \cdot loc \geq \phi \cdot imp$$

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.

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

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.

scenario_config.yaml - 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
<pre>solving: solver: name: highs #(1)! options: highs-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)!</pre>

```
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 cloud computing service provided by [Open Energy Transition](#) Organisation. 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 optimisation 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 optimisation. 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 optimisation process). The value does not matter.

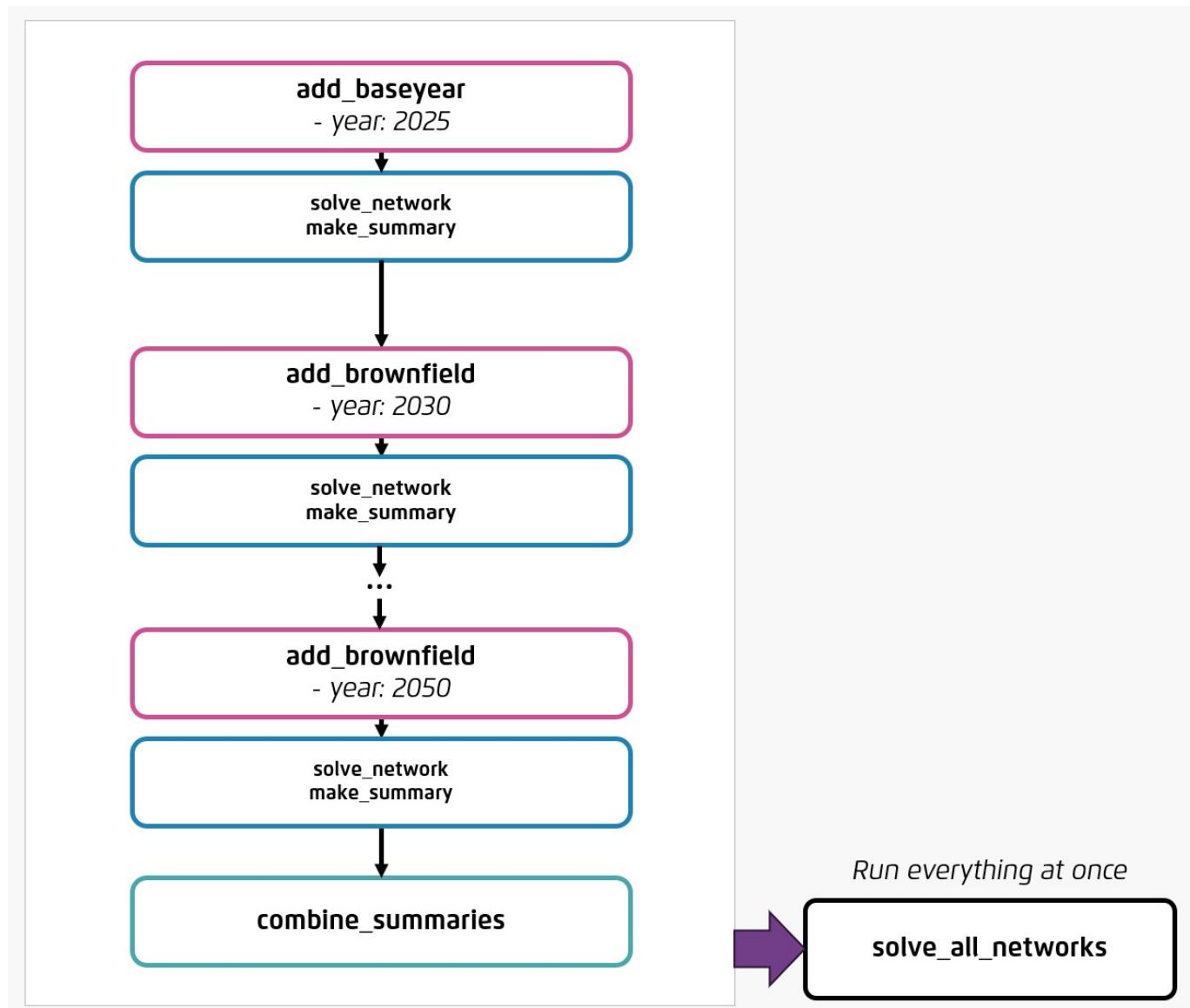
28. Number of CPU threads to be used by the solver for parallel computation to speed up solving time.
29. This parameter changes the algorithm and accepts an integer from `0` to `6`, where `0` denotes automatic choice of the algorithm, `1` is for primal simplex, `2` is for dual simplex, and `4` is for 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 optimisation 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 optimisation 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_basyear	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 <code>config.yaml</code> (see new-model or model-builder-configuration).
add_brownfield	Snakemake rule	Imports all output data from the previous year, using the <code>previous_year_outputs</code> and <code>solved_previous_year</code> functions. The definition of the different years is described in <code>config.yaml</code> (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 <code>solve_all_networks</code> 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 <code>add_basyear</code> or <code>add_brownfield</code> . Optimisation settings are defined in <code>config.yaml</code> (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 customise 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 workflows 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 visualisation 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 `output_scenario_name` in `pypsa-spice/config.yaml`).
3. Optional input name of the first scenario (the name is usually other scenarios from the `output_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/
<code>streamlit run pypsa-spice-vis/main.py</code>	
	<code>streamlit run main.py</code>

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 visualisation 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 charging/discharging 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 (OPEX) 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 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](#).

5.1.1 PyPSA-SPICE or PyPSA training materials from Agora

We also provide training and long-term capacity building programmes to get you started. We have partner programmes 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 programme:

- [Basic PyPSA training using green hydrogen production as an example.](#)
- [Training on PyPSA-SPICE](#)

5.2 Data sources used in PyPSA-SPICE

There are 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](#)

5.3 Publication using PyPSA-SPICE model builder

Here is a list of studies that have utilised 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-optimisation analysis perspective	study link	
Kazakhstan's power system 2035: options for development	study link	

6. Contributing

6.1 How to contribute

PyPSA-SPICE is an open-source repository and thus your contributions are welcome and appreciated!

If you are considering using this model builder, please reach out to us at modelling@agora-thinktanks.org. 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. For troubleshooting, please check the [troubleshooting](#) guide for more information.

6.1.1 Code style

If your contributions involve code changes, please follow the steps below to format your code before creating a pull request. This will help us review your changes more efficiently!

1. Fork the repository on GitHub
2. Clone your fork: `git clone https://github.com/<your-username>/pypsa-spice.git`
3. Fetch the upstream tags `git fetch --tags https://github.com/agoenergy/pypsa-spice/pypsa-spice.git`
4. Install with dependencies in editable mode: `pip install -e .[dev]`
5. Setup linter and formatter, e.g `pre-commit install` (see [Pre-commit](#))
6. Open a new branch and write your code
7. Push your changes to your fork and create a pull request on GitHub

6.1.2 Pre-commit

We use [pre-commit](#) to maintain consistent code style and catch common errors before commits. The pre-commit package is included in the `pypsa-spice` environment. To enable automatic formatting before each commit (recommended), run this command once:

```
Initialize pre-commit
```

```
pre-commit install
```

This will automatically check the changes which are staged before you commit them.

6.2 Code of conduct

This Code of Conduct follows the [Contributor Covenant, version 2.1](#).

6.2.1 Our pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, caste, color, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

6.2.2 Our standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

6.2.3 Enforcement responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

6.2.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

6.2.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at modelling@agora-thinktanks.org. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

6.2.6 Enforcement guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary ban

Community impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent ban

Community impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

6.2.7 Attribution

This Code of Conduct is adapted from the [Contributor covenant version 2.1](#).

Community Impact Guidelines were inspired by [Mozilla CoC](#).

For answers to common questions about this code of conduct, see the [FAQ](#).

7. References

7.1 Citing PyPSA-SPICE

Please use the citation below:

Agora Think Tanks (2025): PyPSA-SPICE: PyPSA-based Scenario Planning and Integrated Capacity Expansion

7.1.1 License

Copyright © 2020-2025, [PyPSA-SPICE Developers](#)

PyPSA-SPICE is licensed under the open-source [GNU General Public License v2.0 or later](#) with the following information:

*This programme is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This programme is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the [GNU General Public License v2.0 or later](#) for more details.*

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 and reviewers

This project is developed and maintained by the team at Agora Energiewende. 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

Technische Universität Berlin (TU Berlin)

- Dr. Elisabeth Zeyen

Institute for Climate and Sustainable Cities (ICSC)

- Jephraim Manansala

Transition Zero (TZ)

- Thomas Kouroughli (formerly Agora Energiewende)

7.2.2 Reviewers

Technische Universität Berlin (TU Berlin)

- Dr. Fabian Neumann