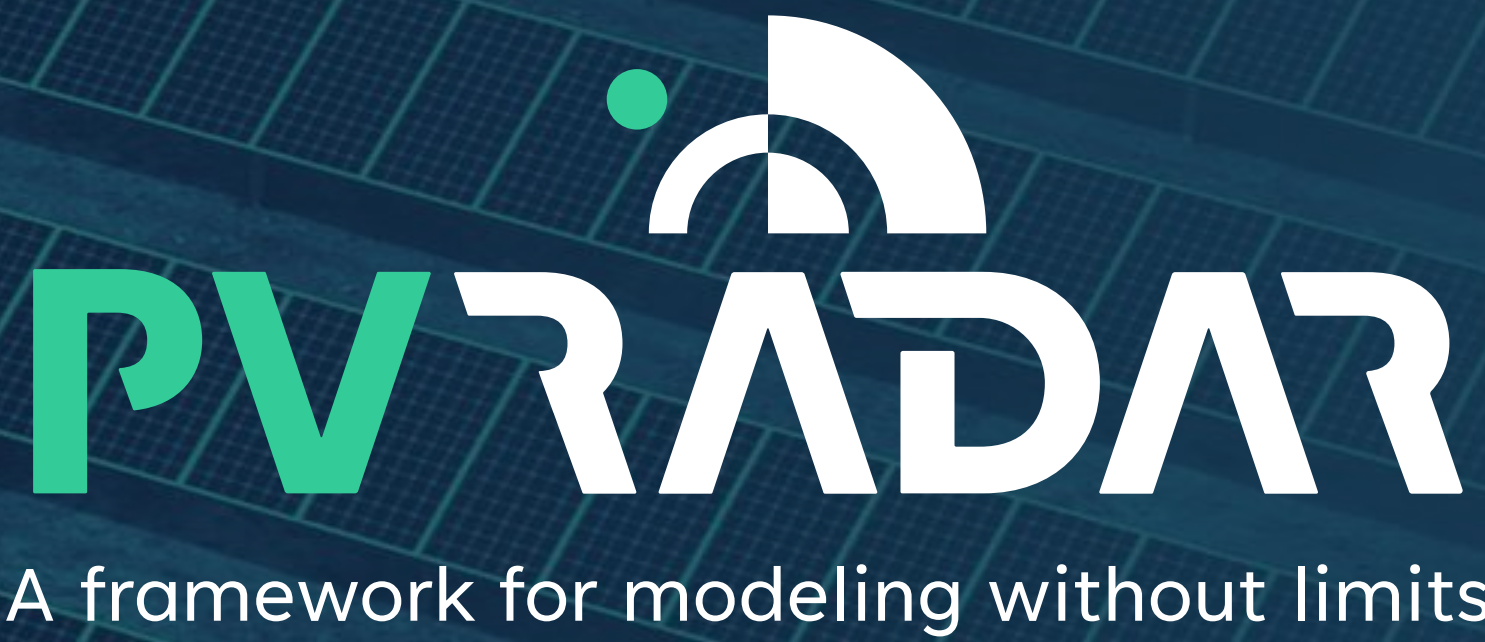


PVRADAR Python Package: Extension to pvlib for faster and easier modeling

Thore Müller, Konstantin Pogorelov, Franco Clandestino
PVRADAR Labs GmbH



Why PVRADAR?

- ✓ Build custom digital twins based on your data.
- ✓ Predict losses and ensure top performance.
- ✓ Open source. Local execution. Full control.

Build the application you need!

- Combine pvlib, our and your own models, as well as measured data to create custom model chains tailored to your need.
- Absolute transparency — everything is open-source, runs locally, and you always know exactly where each result comes from.
- Optimize model parameters to match your data and maximize accuracy.

Seamless data integration

- Access data from ERA5, MERRA2, NOAA, MODIS, and more with a single line of code. Simply specify your preferred format, unit, and time resolution.

```
1 site.resource(R.pm10_volume_concentration(to_unit='g/m^3', datasource='merra2'))
```

- Integrate your internal SCADA or sensor data, or your preferred third-party data, with ease — nothing is shared, and you retain control at all times.
- Freely declare what *resources* you need — required upstream models are automatically executed.

group	resource_type	unit	aggregation_type	datasources	model
irradiance	surface_azimuth_angle	deg	instantaneous		pvlib
irradiance	surface_tilt_angle	deg	instantaneous		pvlib
irradiance	tracker_rotation_angle	deg	instantaneous		pvlib
economics	energy_sales_price	currency/MWh	instantaneous		
economics	revenue	currency	cumulative		
meteo	air_density	kg/m^3	instantaneous	merra2	
meteo	air_temperature	degC	instantaneous	merra2	
meteo	particle_volume_concentration	kg/m^3	instantaneous	merra2	
meteo	pm2_5_volume_concentration	kg/m^3	instantaneous	merra2	
meteo	pm10_volume_concentration	kg/m^3	instantaneous	merra2	
meteo	relative_humidity	%	instantaneous	merra2, noaa	
meteo	wind_speed	m/s	instantaneous	pvgis, merra2	
precipitation	rainfall	mm	cumulative	saemet-grid, era5	
precipitation	snow_density	kg/m^3	instantaneous	era5	
precipitation	snow_depth	mm	instantaneous	merra2, era5	
precipitation	snowfall	mm	cumulative	merra2, era5	
losses	reflection_loss_factor	fraction	instantaneous		pvlib
losses	snow_coverage	fraction	instantaneous		pvlib
losses	snow_loss_factor	fraction	instantaneous		no_loss, pvlib
losses	soiling_loss_factor	fraction	instantaneous		no_loss, pvlib, PVRADAR

Excerpt from the **Resource DB** – available on pvradar.com – which clearly defines every available resource type, its default unit, aggregation type, available data sources, and models. PVRADAR is constantly adding new resource types and data sources.

Start using for free

- install with: `pip install pvradar-sdk`
- API keys for data access are available on our website.
- Contact us for a demo and to learn more.

Example: Fit the HSU model to a measurement dataset and use it to estimate grid-level energy losses.

Start by defining your model as a simple Python function in a jupyter notebook.

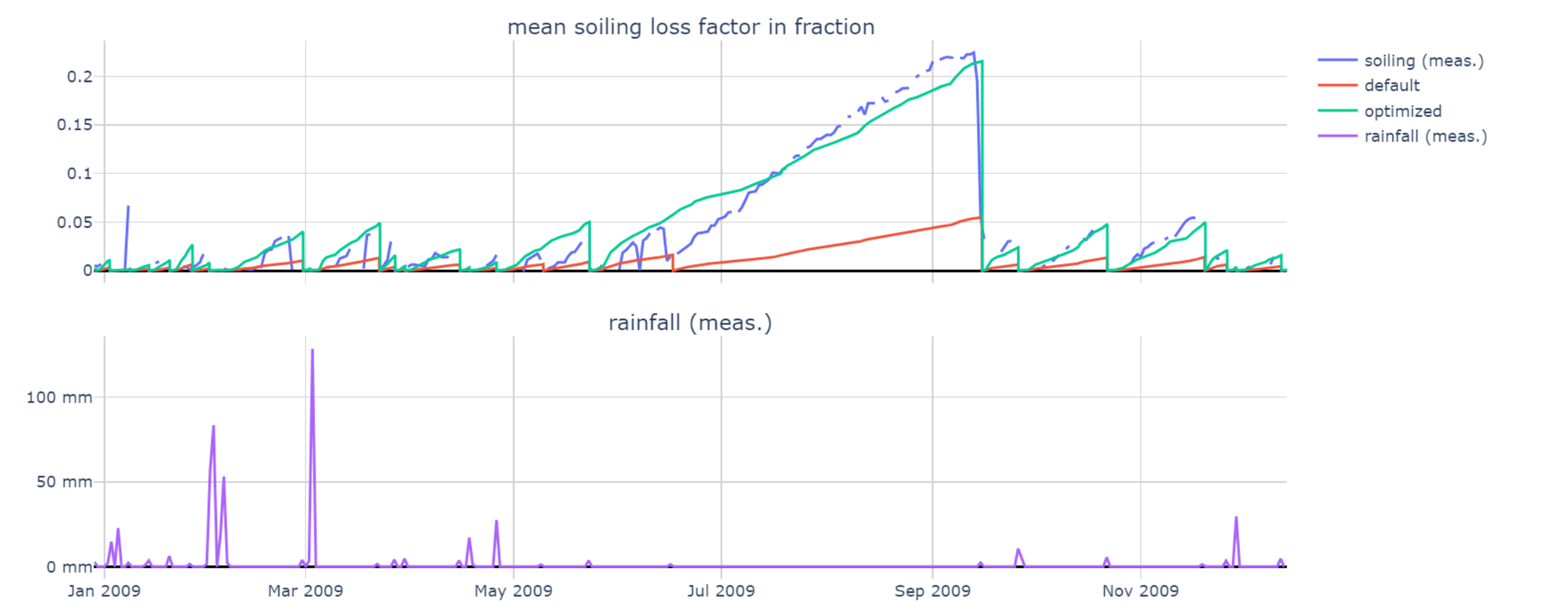
```
1 from pvradar.sdk import R, PvradarSite, resource_plot, resource_type, make_tracker_design, make_fixed_design, for_resource
2
3 @resource_type(R.soiling_loss_factor(set_unit='fraction'))
4 def my_soiling_model(
5     pm2_5: Annotated[pd.Series, R.pm2_5_volume_concentration(to_unit='g/m^3', to_freq='h')],
6     pm10: Annotated[pd.Series, R.pm10_volume_concentration(to_unit='g/m^3', to_freq='h')],
7     rainfall: Annotated[pd.Series, R.rainfall(to_unit='mm', to_freq='h')],
8     tilt_angle: Annotated[pd.Series, R.surface_tilt_angle],
9     daily_cleaning_threshold: Annotated[float, Field(gt=0, lt=10)] = 1,
10    pm2_5_depo_veloc: Annotated[float, Field(gt=0.0001)] = 0.0009,
11    pm10_depo_veloc: Annotated[float, Field(gt=0.001)] = 0.004,
12) -> pd.Series:
13    depo_veloc = ['2.5': pm2_5_depo_veloc, '10': pm10_depo_veloc]
14    soiling_ratio = pvlib.soiling.hsu(
15        rainfall=rainfall,
16        cleaning_threshold=daily_cleaning_threshold,
17        surface_tilt_angle=tilt_angle,
18        pm2_5_depo_veloc=depo_veloc,
19        pm10_depo_veloc=depo_veloc,
20        rain_accum_period=pd.Timedelta('1d'),
21    )
22    return 1 - soiling_ratio
```

Run your model for any location worldwide – PVRADAR automatically imports or calculates the required inputs, so you don't have to worry about sourcing or formatting data.

```
1 site = PvradarSite(location=(36.715, -4.492), interval='2008-12-29..2009-12-13')
2 site.design = make_fixed_design(tilt=30)
3
4 default_estimate = site.run(my_soiling_model,
5                             rainfall=my_rainfall_measurement,
6                             label='default')
```

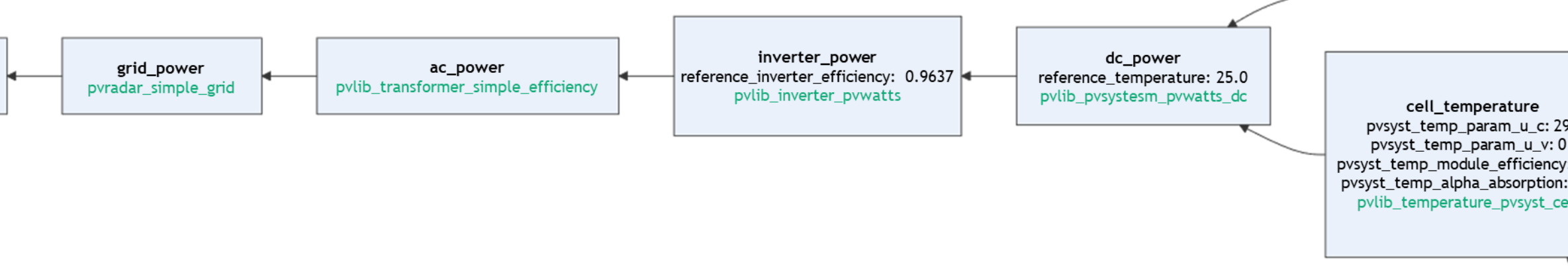
Optimize model parameters to fit your measurements:

```
1 opt_parameters = site.optimize(
2     my_soiling_model,
3     target=my_soiling_measurement,
4     rainfall=my_rainfall_measurement,
5 )
6 print(opt_parameters)
7
8 {'daily_cleaning_threshold': 1.301272089595023, 'pm2_5_depo_veloc': 0.00010000000000000002, 'pm10_depo_veloc': 0.022269065761564996}
```



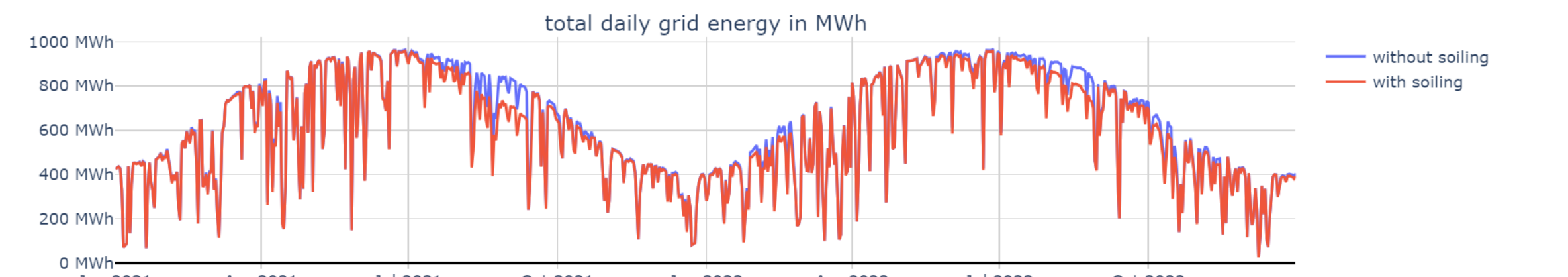
You can always generate a flowchart to visualize exactly how a result is calculated.

```
1 with site.hooks(for_resource(R.soiling_loss_factor).use_model(my_soiling_model, **opt_parameters)):
2     site.profile(R.grid_energy).display_flowchart(show_model_name=True, render='krokodil_svg')
```



Apply the optimized model to calculate energy losses for a nearby PV power plant.

```
1 project = PvradarSite(location=(36.715, -4.492), interval='2021..2022')
2 project.design = make_tracker_design(max_tracking_angle=60, night_stow_angle=15, dc_ac_ratio=1.2)
3
4 grid_energy_without_soiling = project.resource(R.grid_energy(to_unit='MWh', to_freq='D'), label='without soiling')
5
6 with project.hooks(for_resource(R.soiling_loss_factor).use_model(my_soiling_model, **opt_parameters)):
7     grid_energy_with_soiling = project.resource(R.grid_energy(to_unit='MWh', to_freq='D'), label='with soiling')
8
9 resource_plot(grid_energy_without_soiling, grid_energy_with_soiling)
```



Define the **resource type** of the result of your model and specify its unit so it can be correctly processed downstream.

Use the **R-notation** to describe the required inputs in the function header — typically time-series data — including their units and time resolution.

Declare model parameters and set their bounds for optimization.

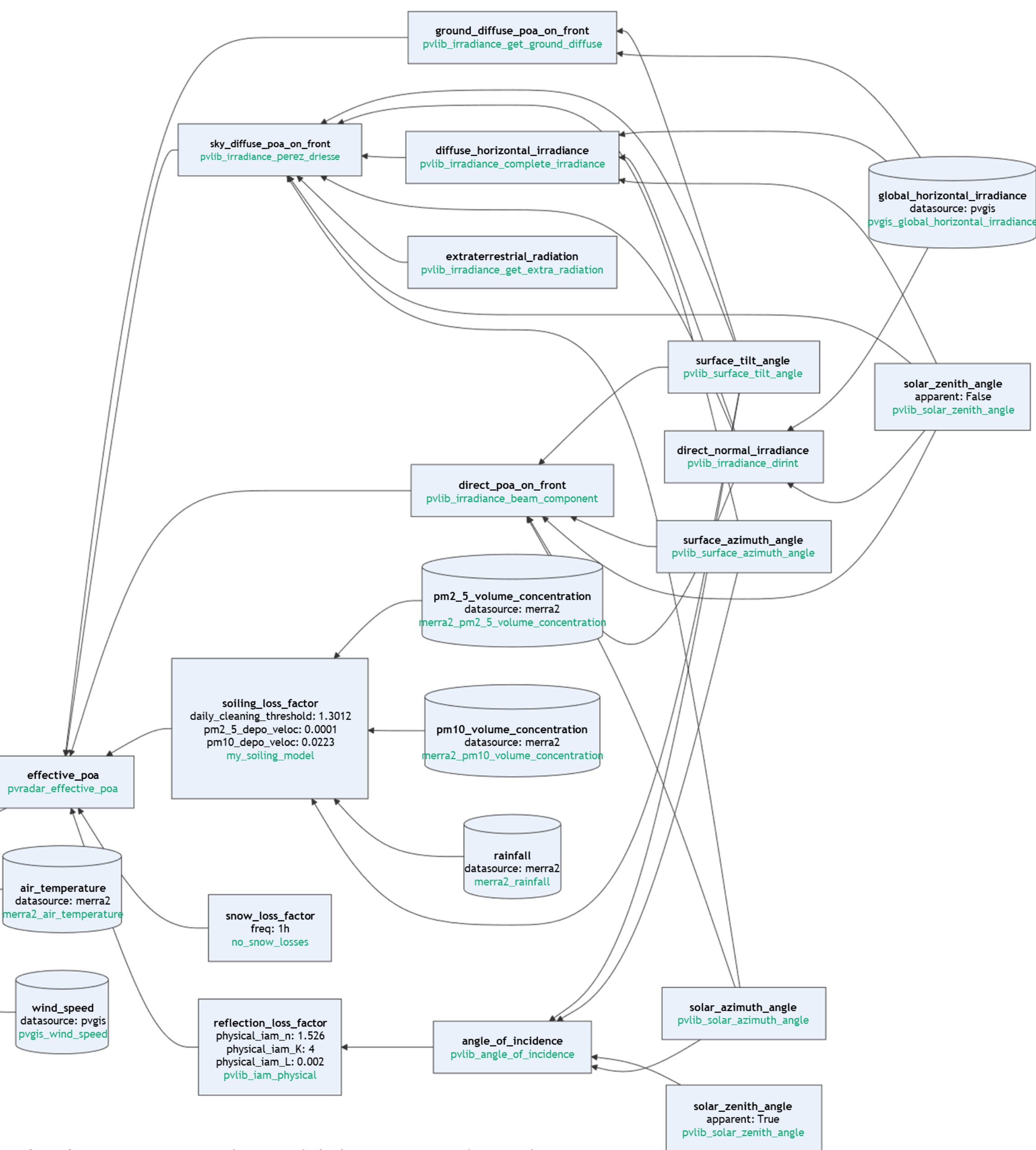
Use the function body to define your model — either based on pvlib or entirely from scratch.

💡 The package includes built-in wrappers for all pvlib models, making them easy to use. PVRADAR Labs also offers additional libraries for cleaning optimization, battery modeling, and custom developments.

The **modeling context**—*site*—is an object that contains information about its location and modeling interval. It manages all data imports from preferred sources or runs models to generate the required inputs.

It can also optimize model parameters to match a target output, such as a measurement.

💡 If you don't have the right validation data, we can help. For soiling alone, our measurement database already includes over 100 measurements from locations around the world.



Use **hooks** to customize the model chain: any node can be replaced by a model, a measurement or a data-import from your preferred data source.

💡 Other use-cases:

- Quantify tracking losses: Replace the tracking angle model with measured tracker angles.
- Compare irradiance datasets: Swap between satellite and ground-station data.
- Estimate sub-hourly clipping losses: Run the model chain in hourly and minute-level resolution.

...and countless other scenarios