

#EU-PVPMC2025

October 29th, 2025

Constructing and Validating pvlib Model Chains for Accurate Site-Specific PV Simulation



THORE MÜLLER
PVRADAR Labs



INTRODUCTION: 2021 PVPMC BLIND MODEL COMPARISON

- Organized by Sandia PVPMC in 2021
- Goal: understand accuracy and variability
- Asked to simulate POA irradiance, module temperature, and DC power
- Weather and system data for six PV systems (Albuquerque & Roskilde)
- Total of 29 submissions, mostly using commercial software

Let's build our own model!



+

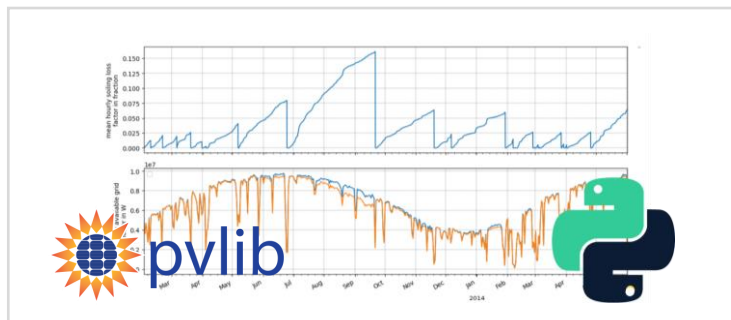


PVRADAR MODELING FRAMEWORK

SDK

Complex modeling made easy

Run pvlib models and build model chains with a single line of code



- ✓ Seamless unit conversion, resampling, and timestamp alignment
- ✓ Create custom models and execute them for any location worldwide

Data Depot

Access data in seconds

Retrieve ready-to-use historic data from satellites and meteo stations

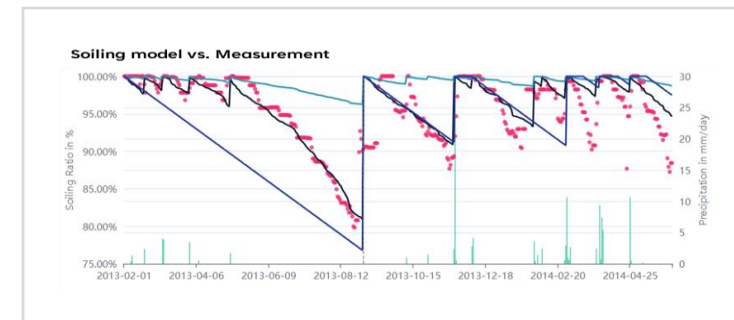


- ✓ Import from dozens of sources with a single line of code
- ✓ Extrapolate historic data for future scenario analysis.

Measurement DB

Achieve maximum accuracy

Adapt model parameters to the unique conditions of your project



- ✓ Validate data sources and optimize models using your own measurements
- ✓ Access soiling and irradiance data from sites worldwide.

DEFINING A MODEL IN PVRADAR

HEADER

```
@standard_resource_type(R.dc_power, override_unit=True)
@synchronize_freq('default')

def pvlib_pvsystem_pvwatts_dc(

    effective_poa: Annotated[pd.Series, R.effective_poa(to_unit='W/m^2')],
    cell_temp: Annotated[pd.Series, R.cell_temperature(to_unit='degC')],

    rated_dc_power: Annotated[float, LambdaArgument(ArrayDesign, lambda d: d.rated_dc_power)],
    gamma_pdc: Annotated[float, LambdaArgument(ModuleDesign, lambda d: d.temperature_coefficient_power)],
    module_power: Annotated[float, LambdaArgument(ModuleDesign, lambda d: d.rated_power)],

    ref_temp: float = 25.0,

) -> pd.Series:
```

BODY

```
    power_one_module = pvlib.pvsystem.pvwatts_dc(
        effective_irradiance=effective_poa,
        temp_cell=cell_temp,
        pdc0=module_power,
        gamma_pdc=gamma_pdc,
        temp_ref=ref_temp,
    )
    dc_power = power_one_module / module_power * rated_dc_power

    return dc_power
```

DEFINING A MODEL IN PVRADAR

HEADER

```
@standard_resource_type(R.dc_power, override_unit=True)
@synchronize_freq('default')

def pvlib_pvsystem_pvwatts_dc(

    effective_poa: Annotated[pd.Series, R.effective_poa(to_unit='W/m^2')],
    cell_temp: Annotated[pd.Series, R.cell_temperature(to_unit='degC')],

    rated_dc_power: Annotated[float, LambdaArgument(ArrayDesign, lambda d: d.rated_dc_power)],
    gamma_pdc: Annotated[float, LambdaArgument(ModuleDesign, lambda d: d.temperature_coefficient_power)],
    module_power: Annotated[float, LambdaArgument(ModuleDesign, lambda d: d.rated_power)],

    ref_temp: float = 25.0,

) -> pd.Series:
```

BODY

```
    power_one_module = pvlib.pvsystem.pvwatts_dc(
        effective_irradiance=effective_poa,
        temp_cell=cell_temp,
        pdc0=module_power,
        gamma_pdc=gamma_pdc,
        temp_ref=ref_temp,
    )
    dc_power = power_one_module / module_power * rated_dc_power

    return dc_power
```

Model body: Define your own custom model, for example using pvlib.

Model library: PVRADAR comes with pre-defined wrappers around all pvlib models!

DEFINING A MODEL IN PVRADAR

HEADER

```
@standard_resource_type(R.dc_power, override_unit=True)
@synchronize_freq('default')
```

```
def pvlib_pvsystem_pvwatts_dc(
```

```
    effective_poa: Annotated[pd.Series, R.effective_poa(to_unit='W/m^2')],
    cell_temp: Annotated[pd.Series, R.cell_temperature(to_unit='degC')],
```

```
    rated_dc_power: Annotated[float, LambdaArgument(ArrayDesign, lambda d: d.rated_dc_power)],
    gamma_pdc: Annotated[float, LambdaArgument(ModuleDesign, lambda d: d.temperature_coefficient_power)],
    module_power: Annotated[float, LambdaArgument(ModuleDesign, lambda d: d.rated_power)],

    ref_temp: float = 25.0,
```

```
) -> pd.Series:
```

BODY

```
    power_one_module = pvlib.pvsystem.pvwatts_dc(
        effective_irradiance=effective_poa,
        temp_cell=cell_temp,
        pdc0=module_power,
        gamma_pdc=gamma_pdc,
        temp_ref=ref_temp,
    )
    dc_power = power_one_module / module_power * rated_dc_power

    return dc_power
```

Physical quantities

- ✓ Unit-conversion
- ✓ Time-zone correction
- ✓ Synchronization / Realignment
- ✓ Resampling
- ✓ ...

DEFINING A MODEL IN PVRADAR

HEADER

```
@standard_resource_type(R.dc_power, override_unit=True)
@synchronize_freq('default')

def pvlib_pvsystem_pvwatts_dc(

    effective_poa: Annotated[pd.Series, R.effective_poa(to_unit='W/m^2')],
    cell_temp: Annotated[pd.Series, R.cell_temperature(to_unit='degC')],

    rated_dc_power: Annotated[float, LambdaArgument(ArrayDesign, lambda d: d.rated_dc_power)],
    gamma_pdc: Annotated[float, LambdaArgument(ModuleDesign, lambda d: d.temperature_coefficient_power)],
    module_power: Annotated[float, LambdaArgument(ModuleDesign, lambda d: d.rated_power)],

    ref_temp: float = 25.0,
```

BODY

```
) -> pd.Series:

    power_one_module = pvlib.pvsystem.pvwatts_dc(
        effective_irradiance=effective_poa,
        temp_cell=cell_temp,
        pdc0=module_power,
        gamma_pdc=gamma_pdc,
        temp_ref=ref_temp,
    )
    dc_power = power_one_module / module_power * rated_dc_power

    return dc_power
```

Design parameters are automatically taken from design description: datasheets, .pan files, etc.

DEFINING A MODEL IN PVRADAR

HEADER

```
@standard_resource_type(R.dc_power, override_unit=True)
@synchronize_freq('default')
```

```
def pvlib_pvsystem_pvwatts_dc(
```

```
    effective_poa: Annotated[pd.Series, R.effective_poa(to_unit='W/m^2')],
    cell_temp: Annotated[pd.Series, R.cell_temperature(to_unit='degC')],

    rated_dc_power: Annotated[float, LambdaArgument(ArrayDesign, lambda d: d.rated_dc_power)],
    gamma_pdc: Annotated[float, LambdaArgument(ModuleDesign, lambda d: d.temperature_coefficient_power)],
    module_power: Annotated[float, LambdaArgument(ModuleDesign, lambda d: d.rated_power)],
```

```
    ref_temp: float = 25.0,
```

Model parameters: define defaults and boundaries

```
) -> pd.Series:
```

BODY

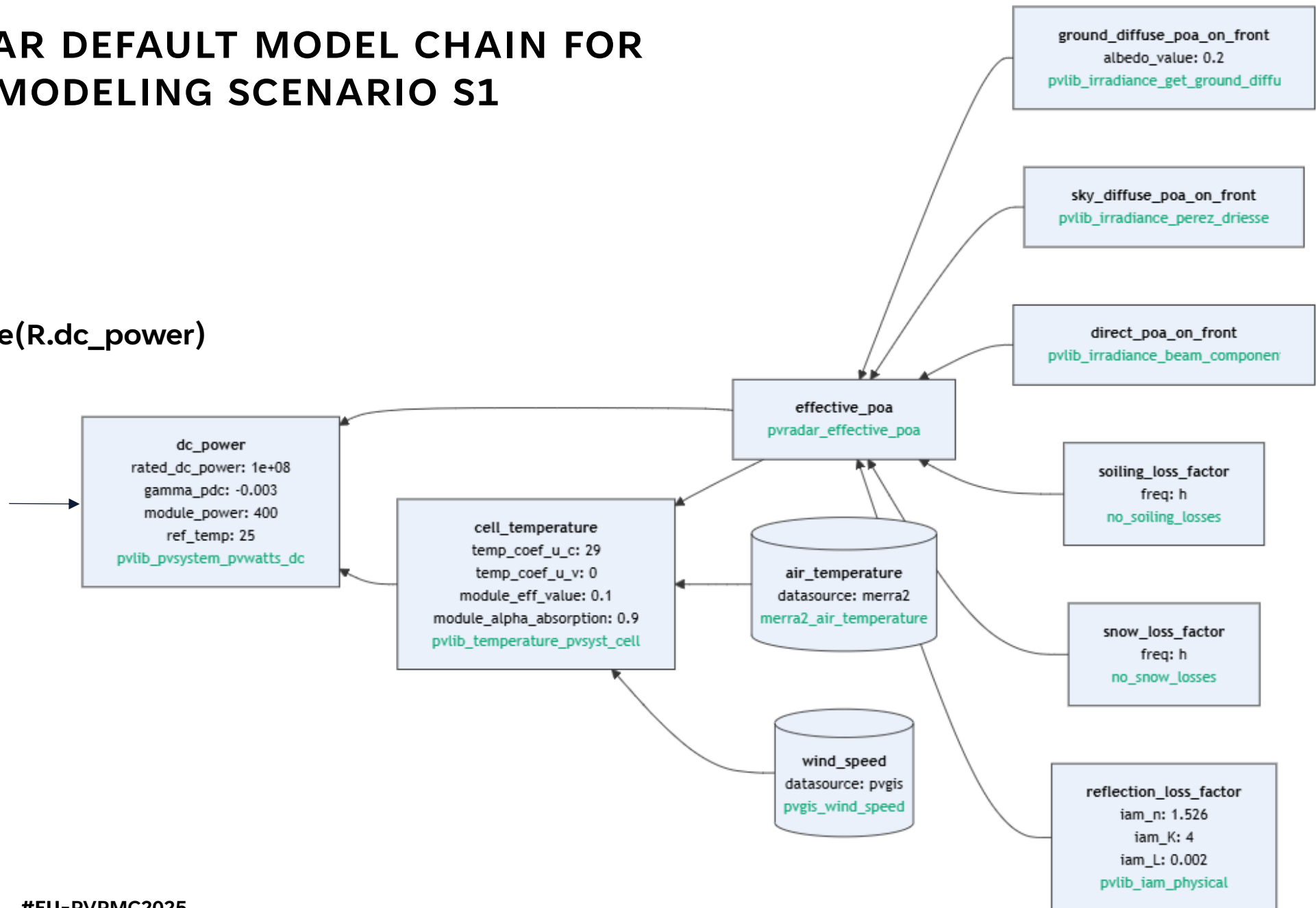
```
    power_one_module = pvlib.pvsystem.pvwatts_dc(
        effective_irradiance=effective_poa,
        temp_cell=cell_temp,
        pdc0=module_power,
        gamma_pdc=gamma_pdc,
        temp_ref=ref_temp,
    )
    dc_power = power_one_module / module_power * rated_dc_power

    return dc_power
```

PVRADAR DEFAULT MODEL CHAIN FOR MODELING SCENARIO S1

site.resource(R.dc_power)

When you
ask for DC
Power:

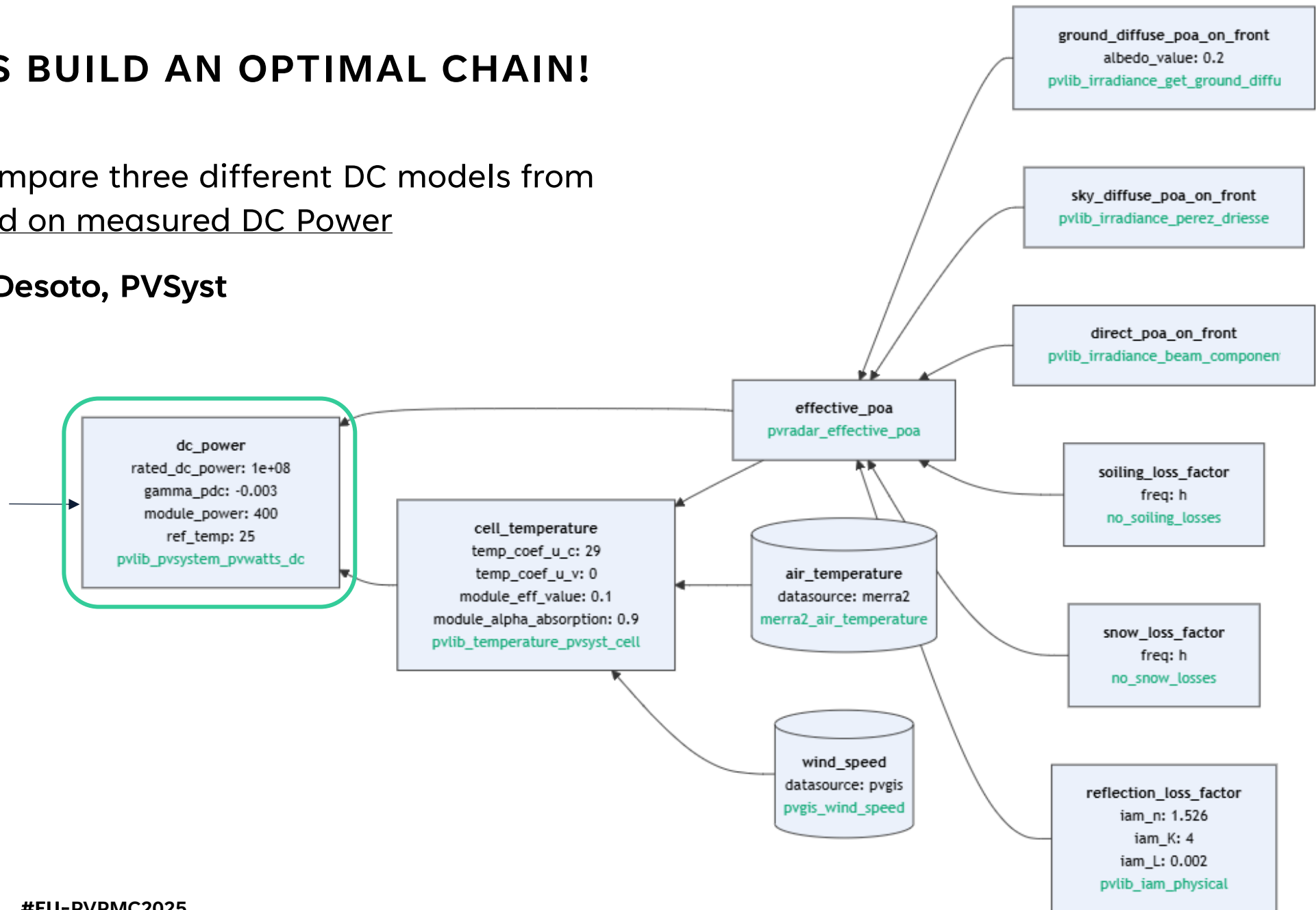


LET'S BUILD AN OPTIMAL CHAIN!

Step 1: Compare three different DC models from pvlib based on measured DC Power

PVWatts, Desoto, PVSyst

When you ask for DC Power:

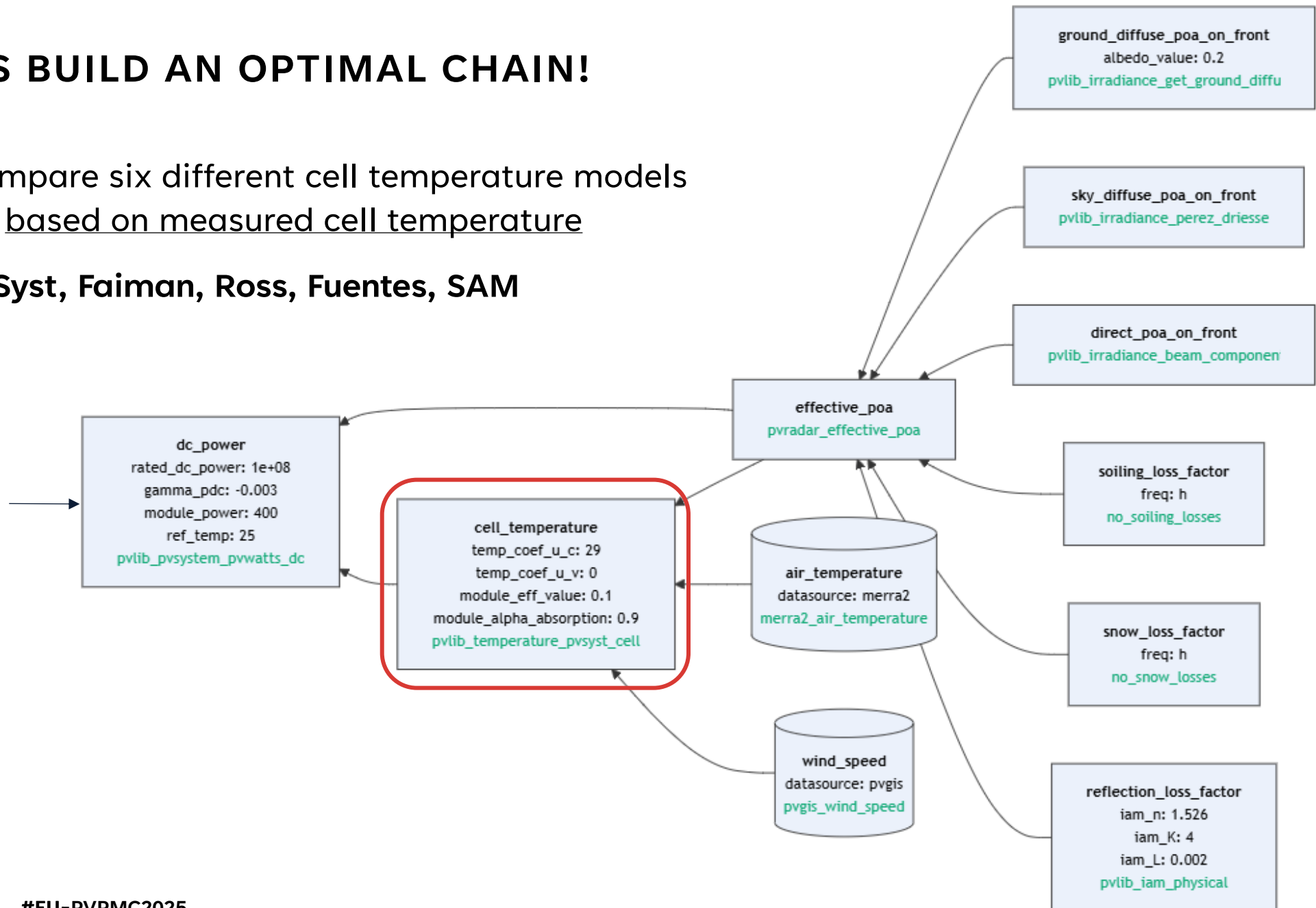


LET'S BUILD AN OPTIMAL CHAIN!

Step 2: Compare six different cell temperature models from pvlib based on measured cell temperature

SAPM, PVSyst, Faiman, Ross, Fuentes, SAM

When you ask for DC Power:

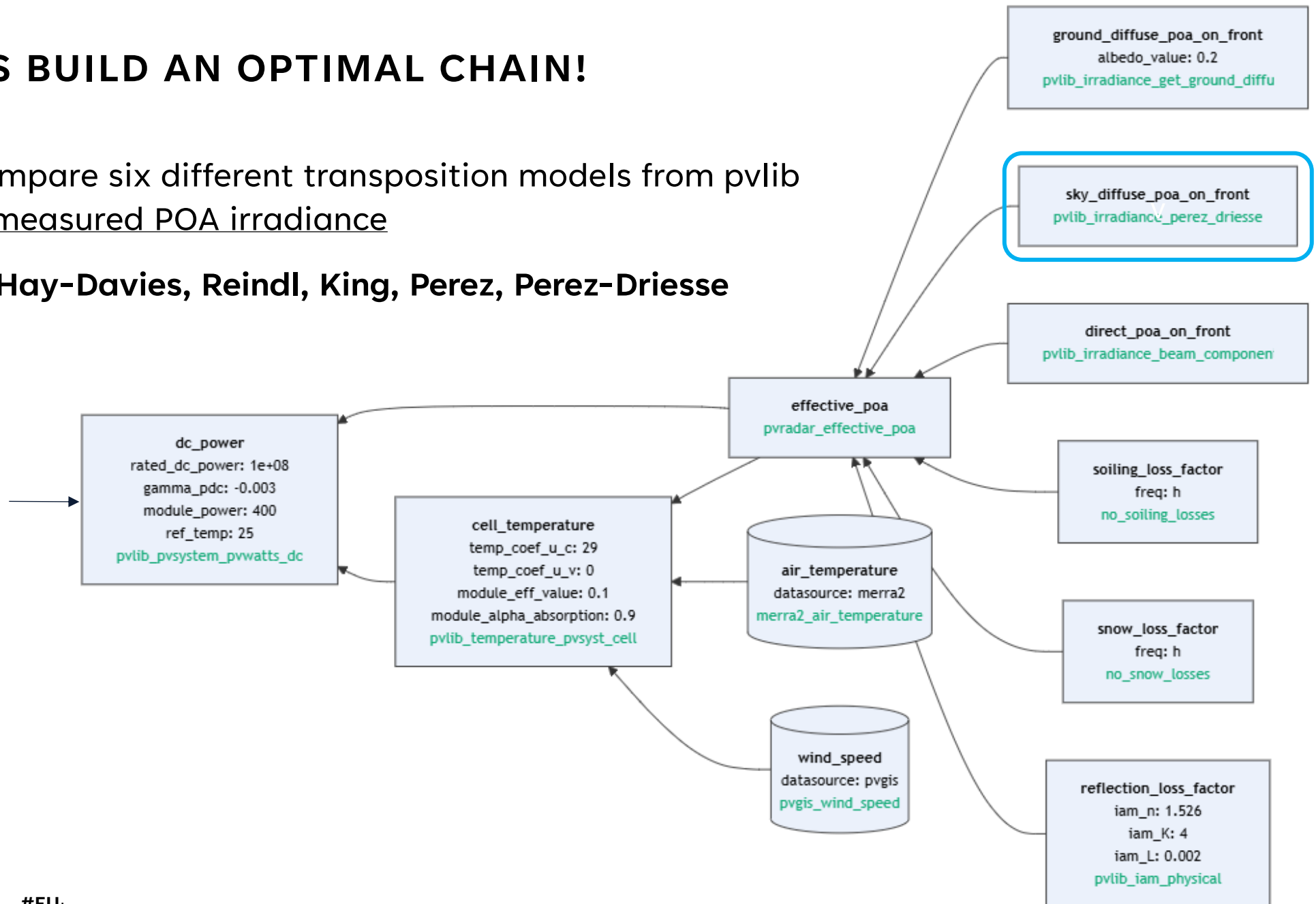


LET'S BUILD AN OPTIMAL CHAIN!

Step 3: Compare six different transposition models from pvlib based on measured POA irradiance

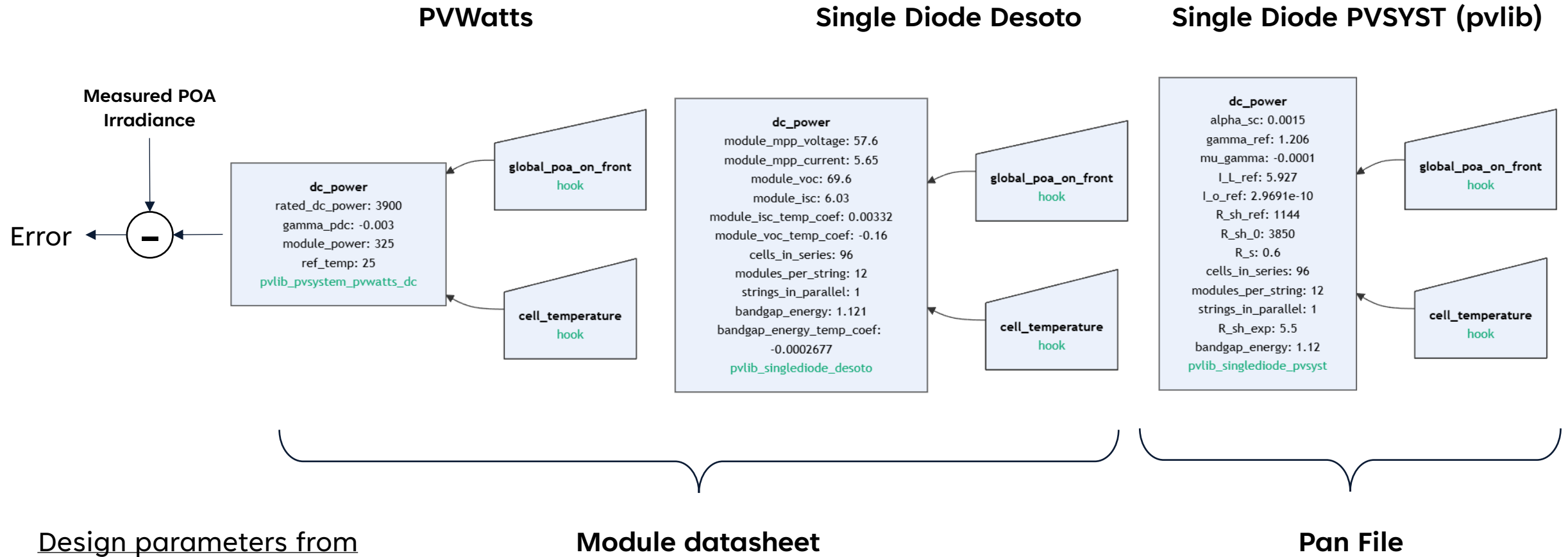
Isotropic, Hay-Davies, Reindl, King, Perez, Perez-Driesse

When you ask for DC Power:

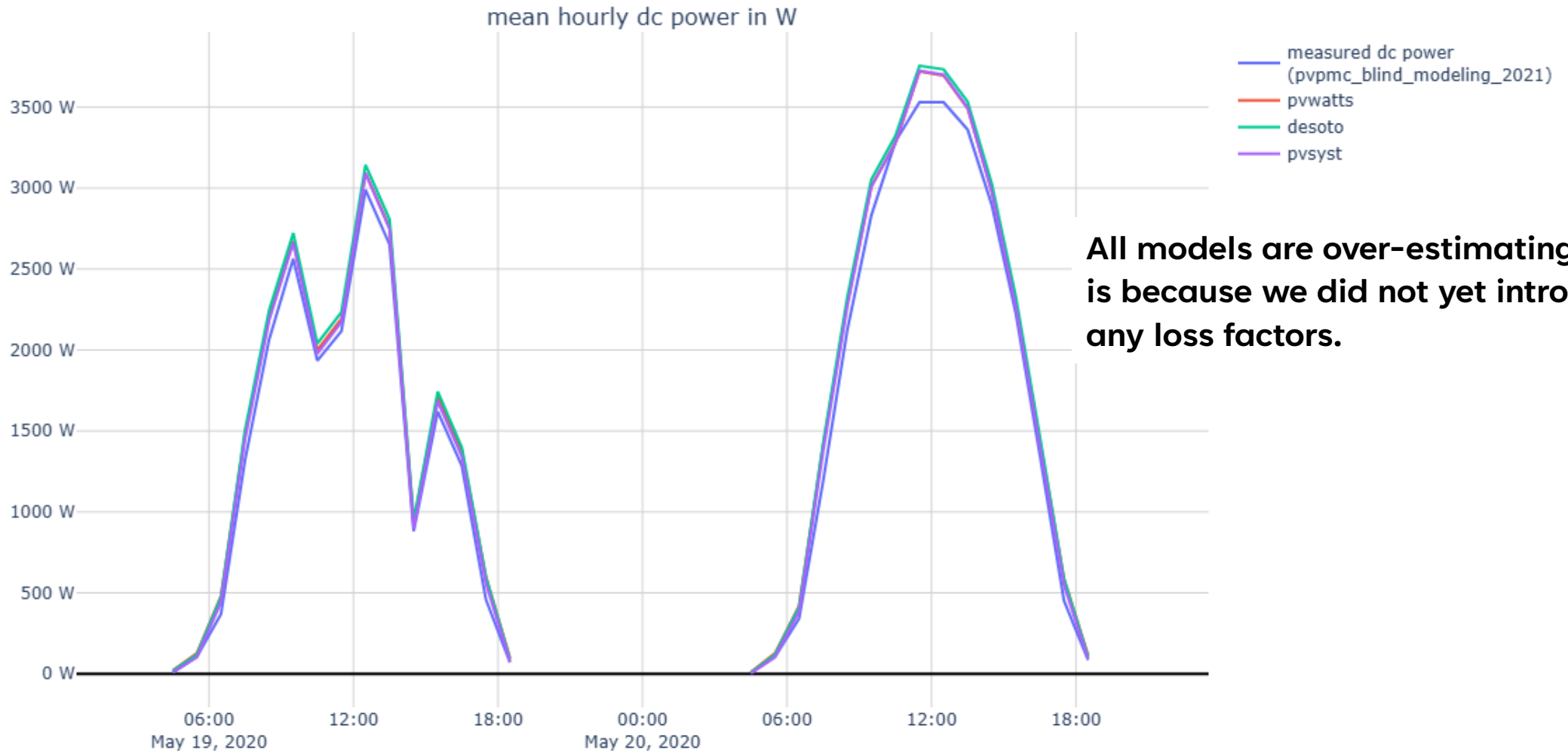


STEP 1: COMPARE THREE DIFFERENT DC MODELS

- PVWATTS, DESOTO, PVSYST -

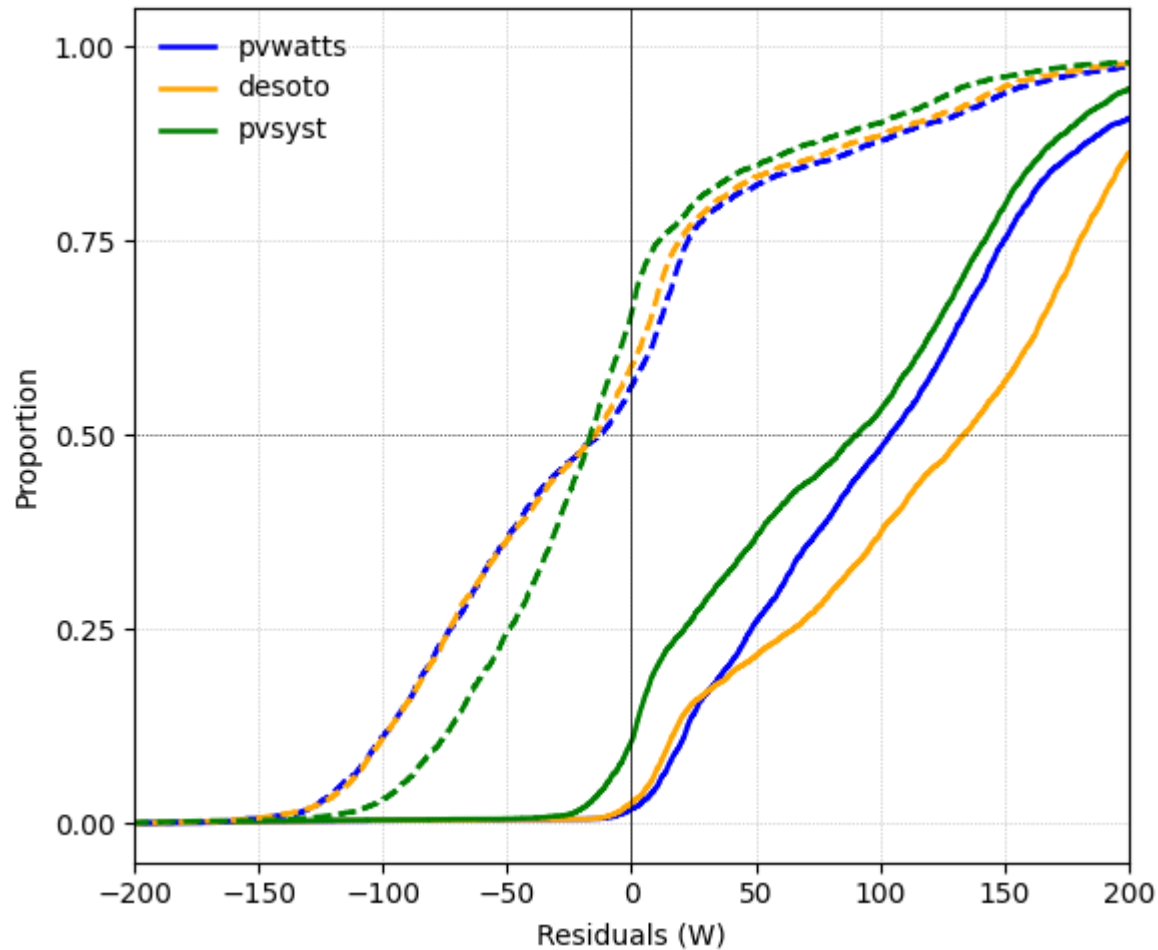


DC MODEL COMPARISON TO MEASURED DC POWER (SCENARIO S1)

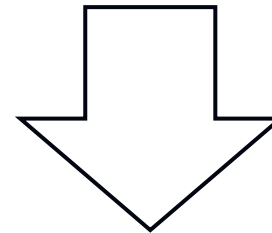


INTRODUCING APPARENT LOSS FACTOR

Cumulative distribution of errors (CDF)



	pvwatts	desoto	pvsyst
MBE [W]	107.49	126.31	89.56
RMSE [W]	134.42	152.69	121.70
Rel. Total Bias [%]	6.11	7.18	5.09



Removing bias:
correcting system losses
and systematic model error

$$P_{DC,corr} = P_{DC} \cdot (1 - L)$$

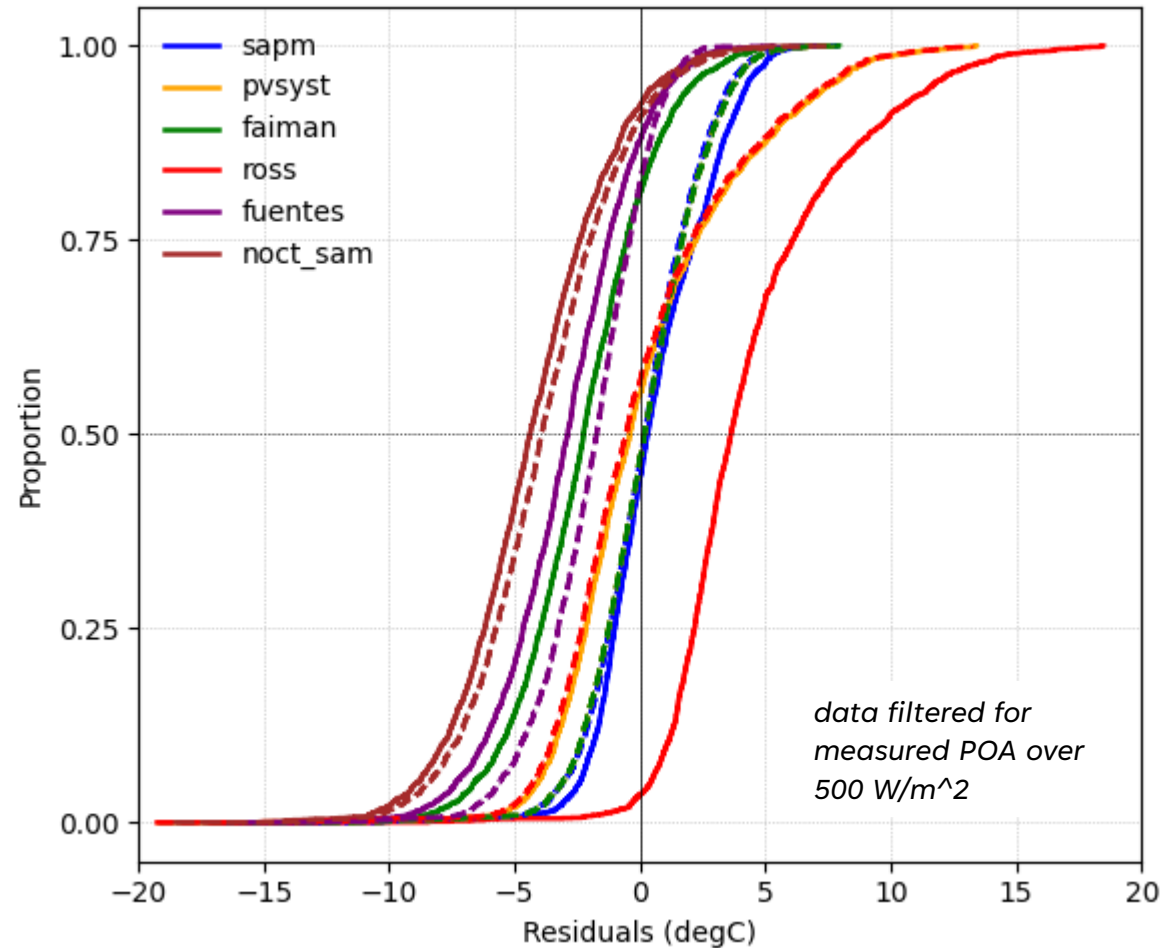
	pvwatts	desoto	pvsyst
MBE [W]	-6.58	-9.08	-4.55
RMSE [W]	93.08	90.47	78.51
Rel. Total Bias [%]	-0.37	-0.52	-0.26

Winner: PVSYST + Single Diode Model

STEP 2: COMPARE SIX DIFFERENT CELL TEMPERATURE MODELS

- SAPM, PVSYST, FAIMAN, ROSS, FUENTES, SAM -

Cumulative distribution of errors (CDF)



	sapm	pvsyst	faiman	ross	fuentes	noct_sam
MBE [degC]	2.45	2.35	0.77	4.84	-0.12	-0.44
RMSE [degC]	3.80	4.32	4.05	5.65	3.73	4.77



Model parameter optimization
only when POA > 500 W/m²

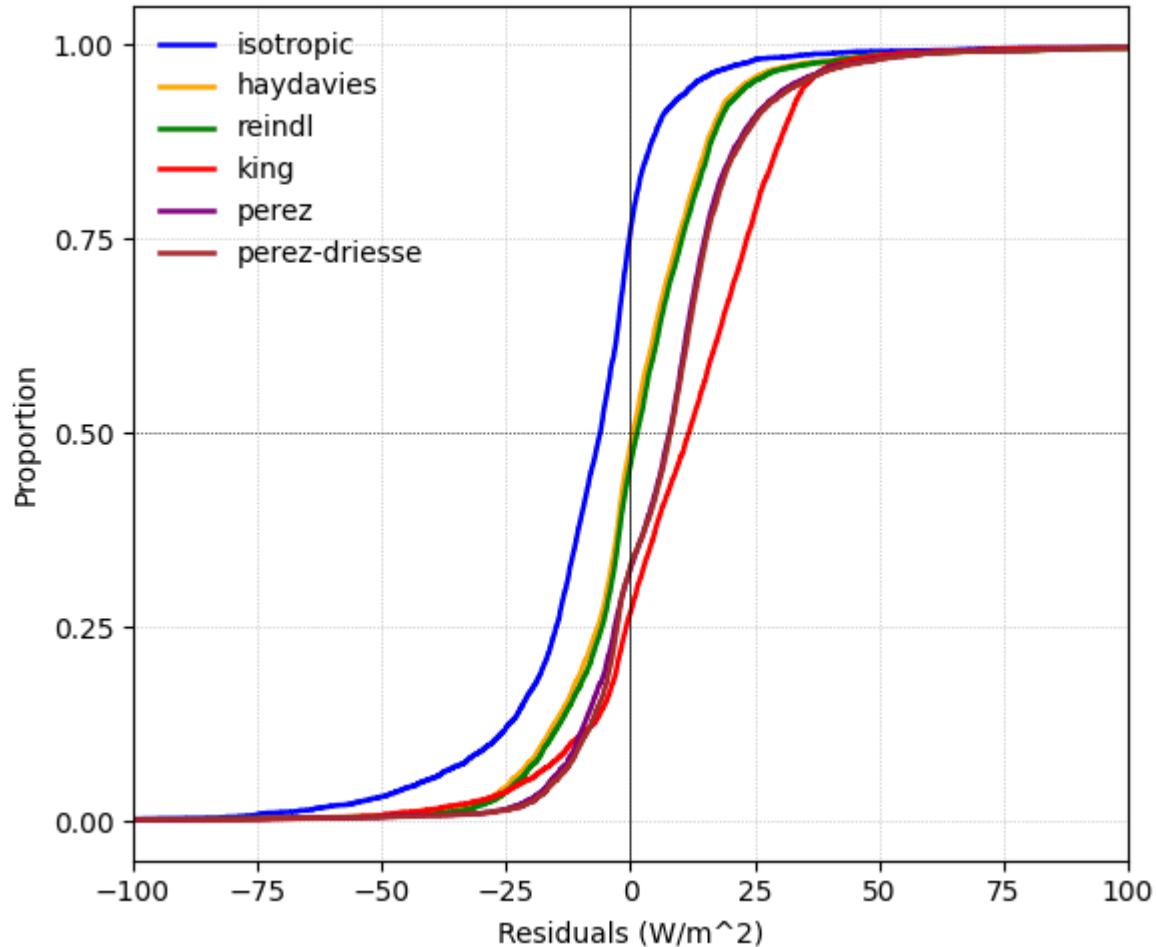
	sapm	pvsyst	faiman	ross	fuentes	noct_sam
MBE [degC]	0.19	0.20	0.20	0.18	-1.94	-3.83
RMSE [degC]	2.16	2.22	2.22	3.66	2.88	4.81

Model	Parameter	Default Value	Optimal Value
sapm	sapm_temp_a	-3.56	-3.40
	sapm_temp_b	-0.075	-0.092
	sapm_deltaT	3	0.0
pvsyst	temp_coef_u_c	29	20.58
	temp_coef_u_v	0	2.75
faiman	temp_coef_u0	25	28.37
	temp_coef_u1	6.84	3.80
ross	module_noct	44.00*	39.86
fuentes	module_emissivity	0.84	1.00
	module_absorption	0.83	0.50
noct_sam	transmittance_absorptance	0.9	1.00

Winner: SAPM + Opt. Parameters

STEP 3: COMPARE SIX DIFFERENT TRANSPOSITION MODELS - ISOTROPIC, HAY-DAVIES, REINDL, KING, PEREZ, PEREZ-DRIESSE -

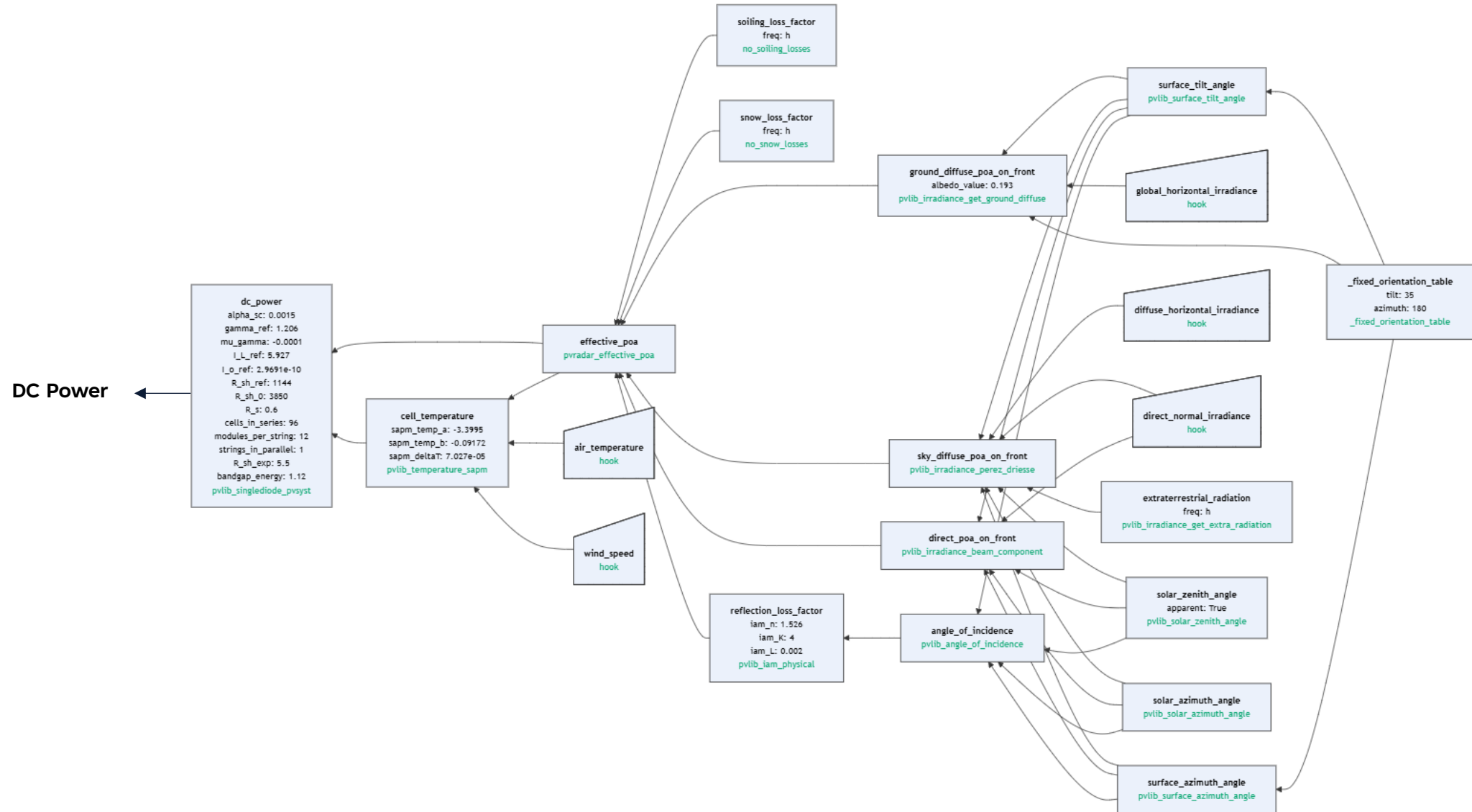
Cumulative distribution of errors (CDF)



	MBE [W/m^2]	RMSE [W/m^2]	Rel. Total Bias [%]
isotropic	-8.34	29.85	-1.67
haydavies	1.60	30.12	0.32
reindl	2.48	30.32	0.50
king	11.01	31.37	2.21
perez	7.69	32.34	1.54
perez-driesse	8.20	31.77	1.65

Winner: **Hay-Davies**

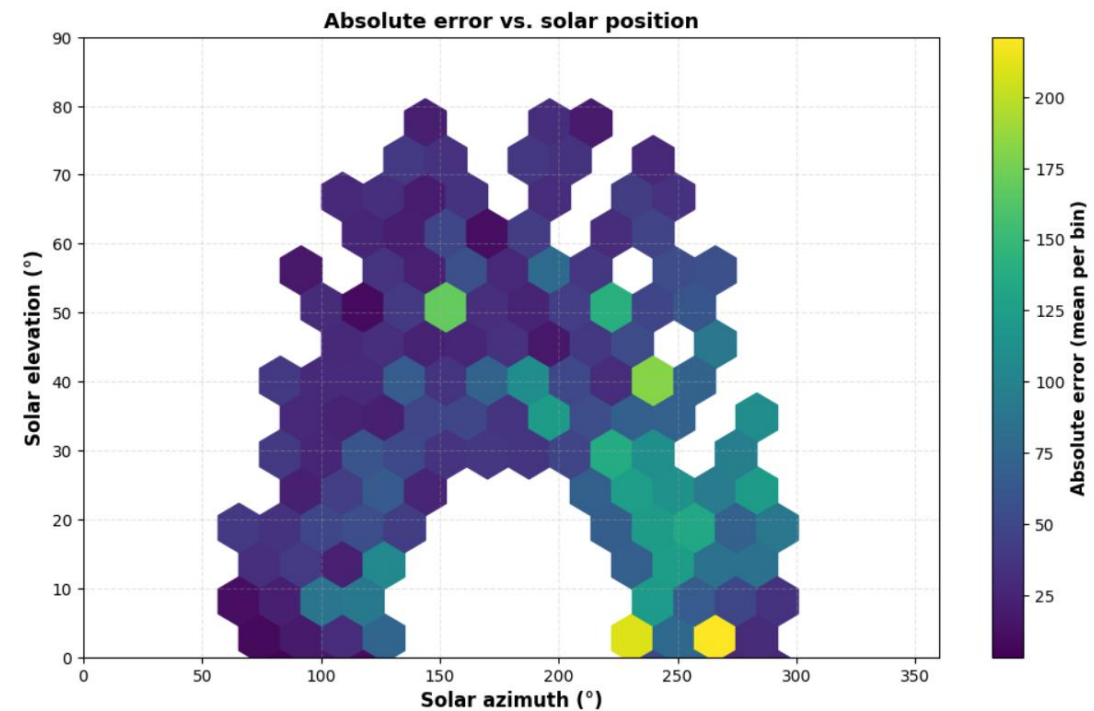
FINAL MODEL CHAIN



RESULTS AND CONCLUSIONS

- ✓ For operating assets: build accurate yield estimation and PR calculation tools
- ✓ Quantify losses: Soiling, snow, tracking error, inverter downtime, derating ...
- ✓ BESS: New model library coming out soon
- ✓ Scale models and analytical workflows across large organizations.
- ✓ Run locally, keep your data safe.

	MBE [W]	RMSE [W]	Rel. Total Bias [%]
Modeled w. meas. POA, CellTemp	-4.55	78.51	-0.26
Modeled w. meas. GHI, DNI, DHI (optimal chain)	-13.72	113.71	-0.78
Modeled w. meas. GHI only (optimal chain)	-29.19	87.43	-1.66



Start Modeling Today

PVRDAR Labs GmbH
Munich, Germany

pvradar.com

thore.mueller@pvradar.com
info@pvradar.com

Base version of PVRADAR is free to use:

[`pip install pvradar-sdk`](#)

Examples available on our website:

www.pvradar.com

To run this analysis on your data:

thore.mueller@pvradar.com

info@pvradar.com