Anemoi-datasets

Baudouin Raoult

https://anemoi-datasets.readthedocs.io

ECMWF

Introduction

Goals

- Create “machine-learning ready” datasets for training data-driven weather forecasts
- Make the loading of data samples as efficient as possible
- Provide rich metadata that can be used in training and inference

Non-goals

- Is not a replacement for the original source of data
- Consider the datasets created by anemoi-datasets is pre-processed data that are as close as possible to the needs of training

Training data-driven forecasts

- A “sample” is the state of the atmosphere a time T

- We want to train a model that, given the state of the atmosphere a time T returns the state of the atmosphere a T+1

- The state of the atmosphere is a collections of meteorological fields (variables)
  - 2m temperature, surface pressure,…
  - geopotential, winds, temperature on 1000 hPa, 500 hPa, …

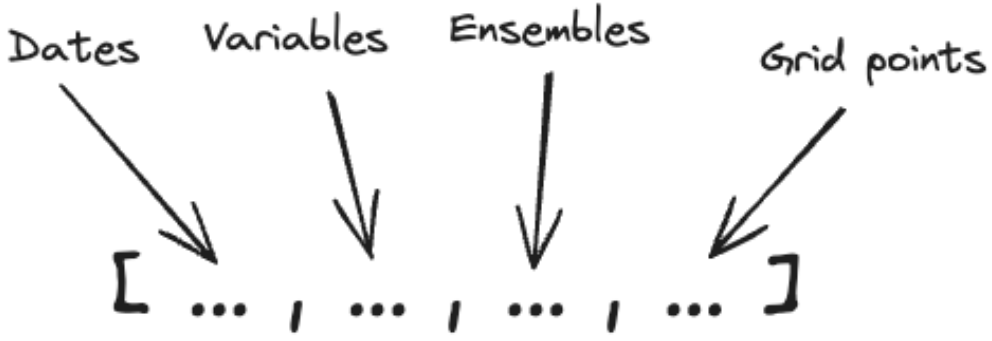
Training a weather forecast

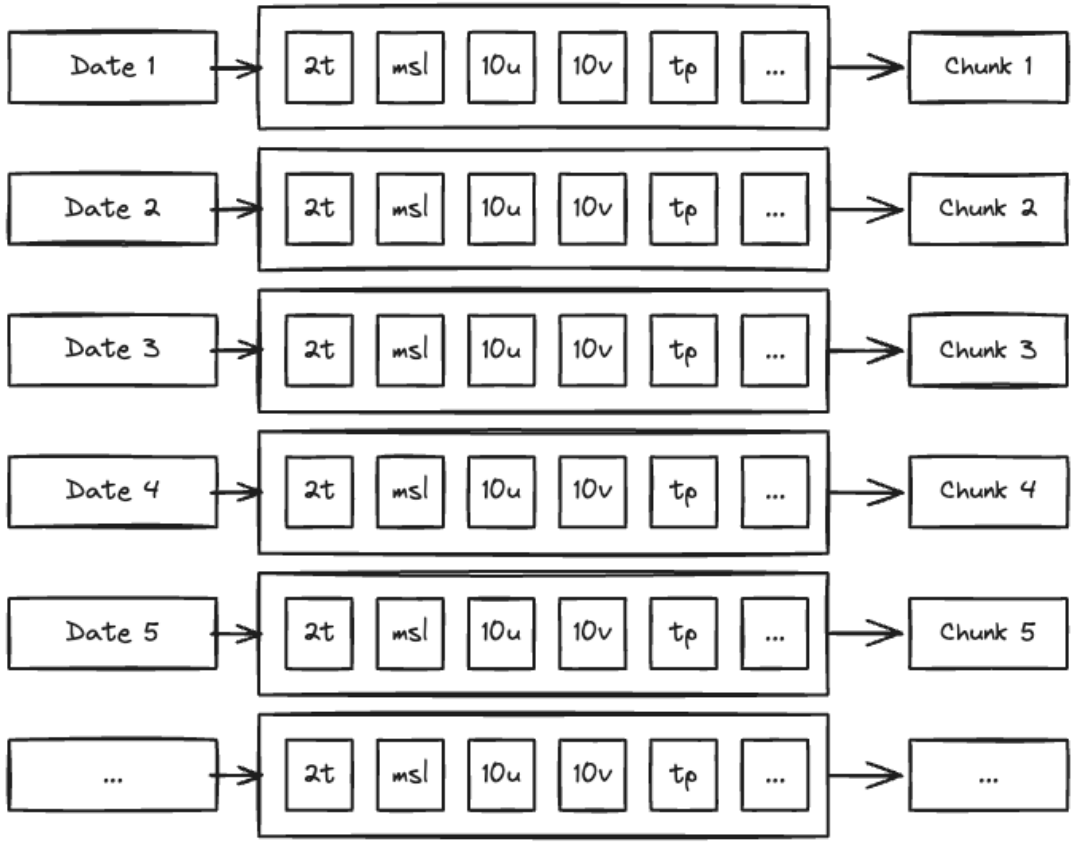
```
x, y = ds[n : n + 1]
y_hat = model.predict(x)
loss = model.loss(y, y_hat)
```

- A dataset like ERA5 is too large to fit in memory
- We need to store the data on a filesystem
- We want to create a dataset where each sample is one I/O

Using Zarr

- The Zarr format is a good fit for the requirements
- It offers an array-like view on a collection of files (chunks)
- We use it to define an array that match exactly a sample
- Each file is a single date





Using datasets

Introduction

To open a dataset, use the `open_dataset` function:

```
from anemoi.datasets import open_dataset

ds = open_dataset("/path/to/dataset.zarr")
```

You can then access the data in the dataset using the `ds` object as if it was a **NumPy** array:

```
print(ds.shape)
print(len(ds))
print(ds[0])
print(ds[10:20])
```

Introduction (cont.)

One of the main features is the ability to subset:

```
from anemoi.datasets import open_dataset

ds = open_dataset("path/to/dataset.zarr",
                  start=2000,
                  end=2020)
```

or combine datasets:

```
from anemoi.datasets import open_dataset

ds = open_dataset("path/to/dataset1.zarr",
                  "path/to/dataset2.zarr")
```

Opening a dataset

```
from anemoi.datasets import open_dataset

ds = open_dataset(dataset,
                  option1=value1,
                  option2=...)
```

or

```
ds = open_dataset({"dataset": dataset,
                   "option1"=value1,
                   "option2"=...})
```

Opening a dataset

```
from anemoi.datasets import open_dataset


ds = open_dataset("/path/to/dataset.zarr")

ds = open_dataset("https://path/to/dataset.zarr")

ds = open_dataset("s3://path/to/dataset.zarr")
```

Open by name (recommended)

```python
from anemoi.datasets import open_dataset

ds = open_dataset("dataset_name")
```

Using `~/.config/anemoi/settings.toml`.

```
[datasets]
path = [
    "/ml-datasets/stable",
    "s3://ml-datasets",
]

[datasets.named]
test = "/home/mlx/test-dataset.zarr”
```

Useful to share code between sites

Re-use an opened dataset

```
from anemoi.datasets import open_dataset

ds1 = open_dataset("/path/to/dataset.zarr")

ds2 = open_dataset(ds1,
                   frequency="24h",
                   start="2000",
                   end="2010")
```

The dictionary can be as complex as needed

```python
from anemoi.datasets import open_dataset

config = {
    "dataset": {
        "ensemble": [
            "/path/to/dataset1.zarr",
            {"dataset": "dataset_name", "end": 2010},
            {"dataset": "s3://path/to/dataset3.zarr",
                        "start": 2000,
                        "end": 2010},
        ],
        "frequency": "24h",
    },
    "select": ["2t", "msl"],
}

ds = open_dataset(config)
```

Common use-case: defined a dataset in a config file

```
with open("config.yaml") as file:
    config = yaml.safe_load(file)

ds = open_dataset(config)
```

Subsetting – (filtering by date)

```
open_dataset(dataset, start=1980)
open_dataset(dataset, end=2020)
open_dataset(dataset, start=1980, end=2020)
```

Useful to create training, test and validation sets from the same input dataset

Subestting

The following are equivalent ways of describing `start` or `end`:

- `2020` and `"2020"`
- `202306`, `"202306"` and `"2023-06"`
- `20200301`, `"20200301"` and `"2020-03-01"`

Subsetting: frequency

You can change the frequency of the dataset by passing a string with:

```
ds = open_dataset(dataset,
     frequency="6h")
```

The new frequency must be a multiple of the original frequency.

To artificially increase the frequency, you can use the `interpolate_frequency` option. This will create new dates in the dataset by linearly interpolating the data values between the original dates.

```
ds = open_dataset(dataset,
                  interpolate_frequency="10m")
```

Selecting variables

```
ds = open_dataset(dataset,
                  select=["2t", "tp"])

ds = open_dataset(dataset,
                  select={"2t", "tp"})

ds = open_dataset(dataset,
                  drop=["10u", "10v"])
```

Reordering variables

… using a list

```
ds = open_dataset(
    dataset,
    reorder=["2t", "msl", "sp", "10u", "10v"],
)
```

… or using a dictionary

```
ds = open_dataset(
    dataset,
    reorder={
        "2t":  0,
        "msl": 1,
        "sp":  2,
        "10u": 3,
        "10v": 4,
    },
)
```

Renaming variables

```
ds = open_dataset(dataset,
       rename={"2t": "t2m"})
```

Useful when combining datasets.

rescale

When combining datasets, you may want to rescale the variables so that their have matching units. This can be done with the rescale option:

```
ds = open_dataset(
    dataset,
    rescale={"2t": {"scale": 1.0, "offset": -273.15}},
)
```

```
ds = open_dataset(
    dataset,
    rescale={"2t": (1.0, -273.15)},
)
```

```
ds = open_dataset(
    dataset,
    rescale={
        "2t": ("K", "degC"),
        "tp": ("m", "mm"),
    },
)
```

The rescale option will also rescale the statistics. The rescaling is currently limited to simple linear conversions.

Selecting ensemble members

Select a single element:

```
ds = open_dataset(
    dataset,
    number=1,
)
```

… or a list:

```
ds = open_dataset(
    dataset,
    number=[1, 3, 5],
)
```

You can also use `member` The difference between the two is that `number` is **1-based**, while `member` is **0-based**.

Combining datasets

```
ds = open_dataset(dataset1, dataset2)
```

```
ds = open_dataset([dataset1, dataset2])
```

The operation depends on the datasets

Combine: concatenate (same variables, dates follow each other)

	2t	msl	sp
1999-01-01			
1999-01-02			
...			
1999-12-31			

	2t	msl	sp
2000-01-01			
2000-01-02			
...			
2000-12-31			

Concatenate

	2t	msl	sp
1999-01-01			
1999-01-02			
...			
...			
...			
...			
...			
...			
2000-12-31			

Combine : join (same range of dates, different variables)

	2t	msl	sp
2000-01-01			
2000-01-02			
...			
2000-12-31			

	10u	10v
2000-01-01		
2000-01-02		
...		
2000-12-31		

Join

	2t	msl	sp	10u	10v
2000-01-01					
2000-01-02					
...					
2000-12-31					

Combine : join (same range of dates, overlapping variables)

	2t	msl	sp
2000-01-01			
2000-01-02			
...			
2000-12-31			

	msl	tp
2000-01-01		
2000-01-02		
...		
2000-12-31		

Join

	2t	msl	sp	tp
2000-01-01				
2000-01-02				
...				
2000-12-31				

Matching attributes

```
ds = open_dataset(
    join=[dataset1, dataset2],
    adjust="frequency",
)
```

```
ds = open_dataset(
    join=[dataset1, dataset2],
    adjust=["start", "end", "frequency"],
)
```

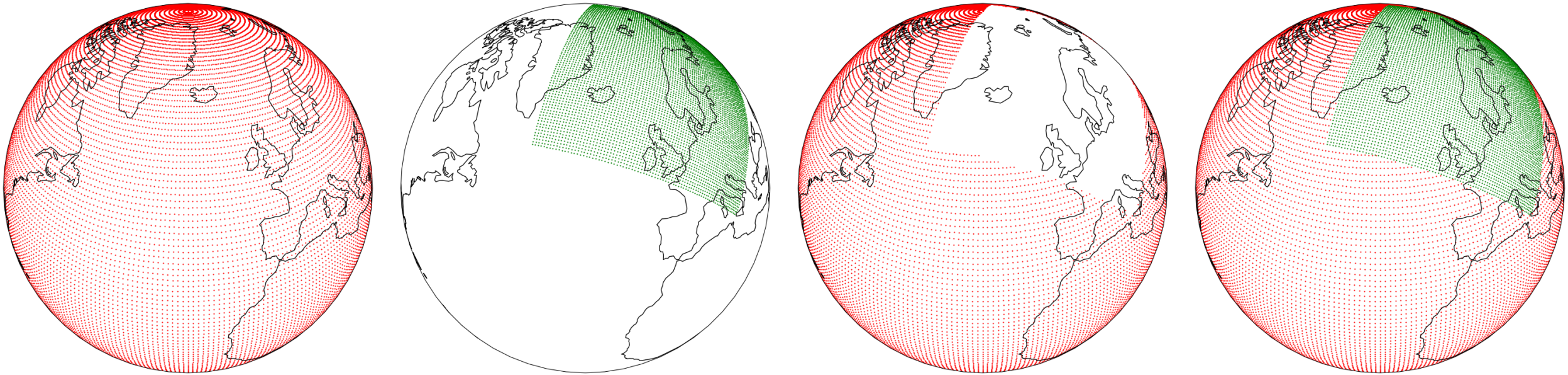
```
ds = open_dataset(join=[dataset1, dataset2], adjust="dates")
```

```
ds = open_dataset(concat=[dataset1, dataset2],ajust="variables")
```

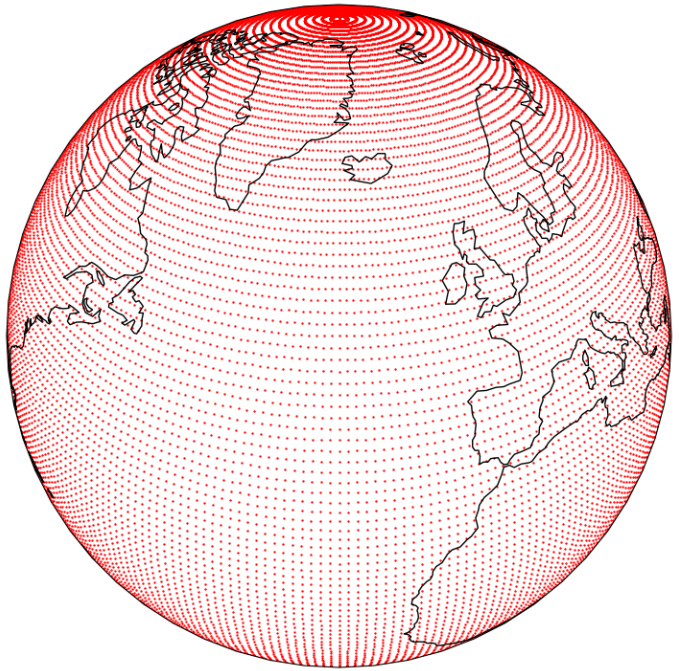
```
ds = open_dataset(
    cutout=[dataset1, dataset2],
    adjust="all",
)
```

Cutout - Use to combine limited are models and global models

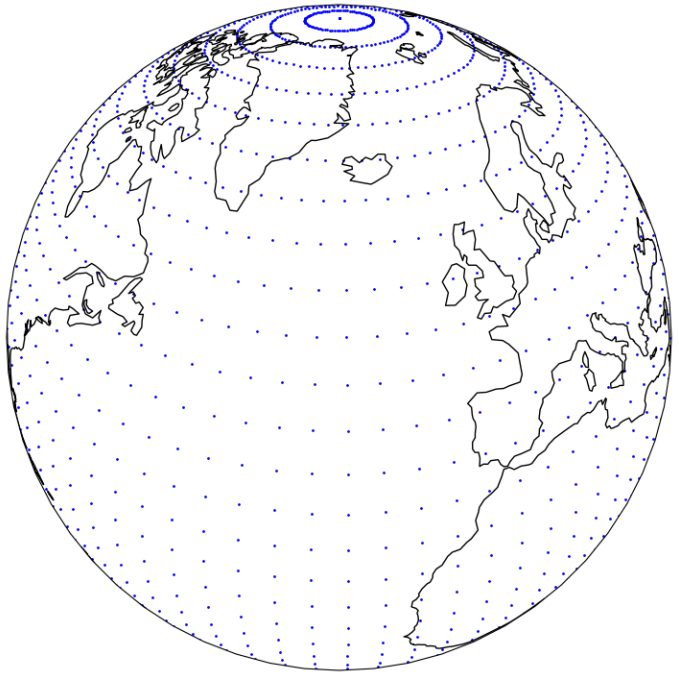
```
from anemoi.datasets import open_dataset

ds = open_dataset(cutout=[lam_dataset,
                          global_dataset])
```



Thinning - Used for quick prototyping

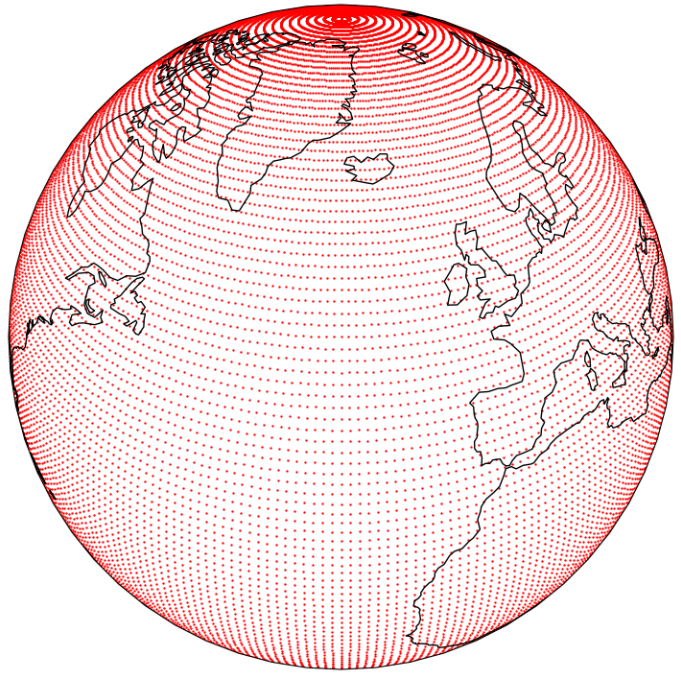


```
ds = open_dataset(dataset,
                  thinning=4)
```

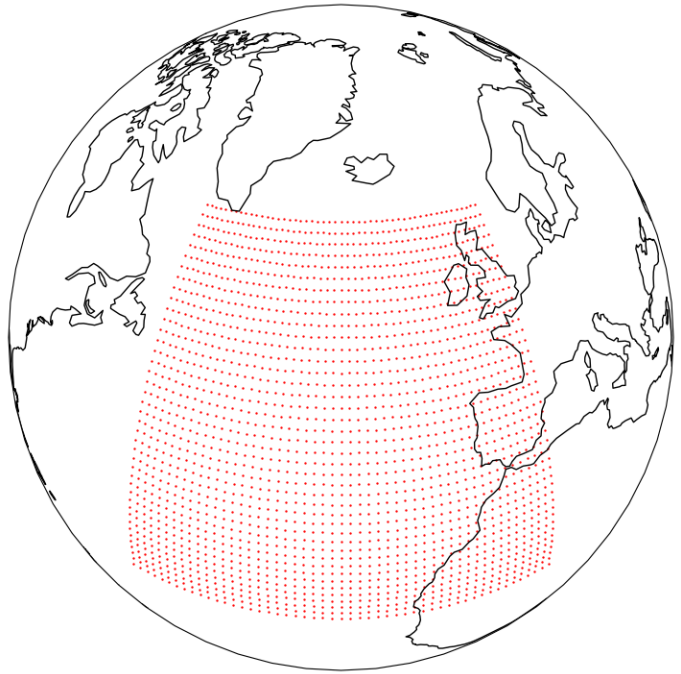


Area

```
ds = open_dataset(dataset,
                  area=(60, -50, 20, 0))
```



```
ds = open_dataset(dataset1,
                  area=dataset2)
```



Indexing – Support (most) Numpy's indexing and slicing

```
ds = open_dataset(…)
print(len(ds))

for n in ds:
  print(n)

print(ds[1])
print(ds[1:10])
print(ds[1:10, 5, :])
```

Each slice is a NumPy array

Warning: `ds[:]` will load the whole dataset in memory

Attributes

Attribute	Description
shape	A tuple of the dataset's dimensions.
field_shape	The original shape of a single field, either 1D or 2D. When building datasets, the fields are flattened to 1D.
dtype	The dataset's NumPy data type.
dates	The dataset's dates, as a NumPy vector of datetime64 objects.
frequency	The dataset's frequency (i.e the delta between two consecutive dates)
latitudes	The dataset's latitudes as a NumPy vector.
longitudes	The dataset's longitudes as a NumPy vector.
statistics	(Next slide)
resolution	The dataset's resolution.
name_to_index	A dictionary mapping variable names to their indices. `print(dataset.name_to_index["2t"])`
variables	A list of the dataset's variable names, in the order they appear in the dataset.
missing	The set of indices of the missing dates.

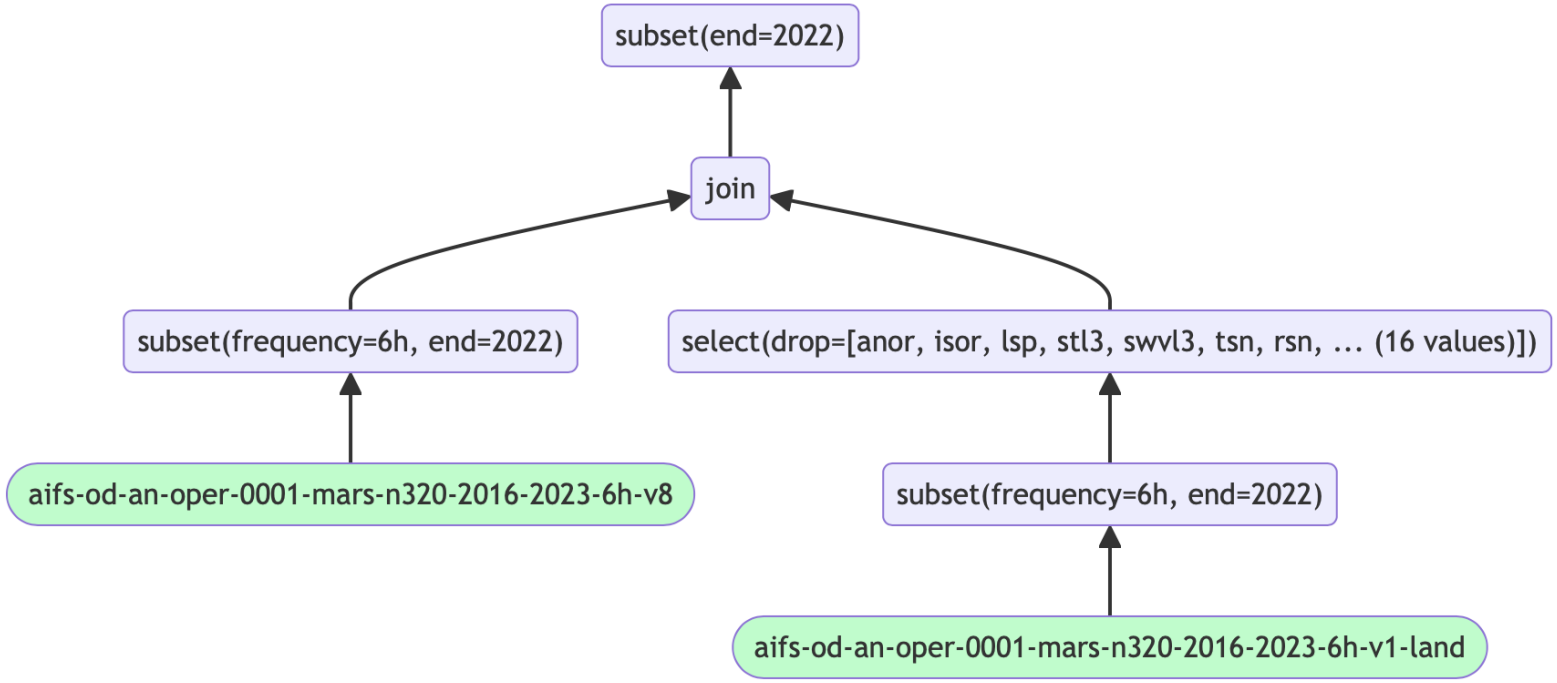
Statistics

```
ds = open_dataset(…)
stats = ds.statistics

stats["mean"]
stats["stdev"]
stats["minimum"]
stats["maximum"]


idx = ds.name_to_index("msl")

print(stats["mean"][idx], stats["stdev"][idx])
```

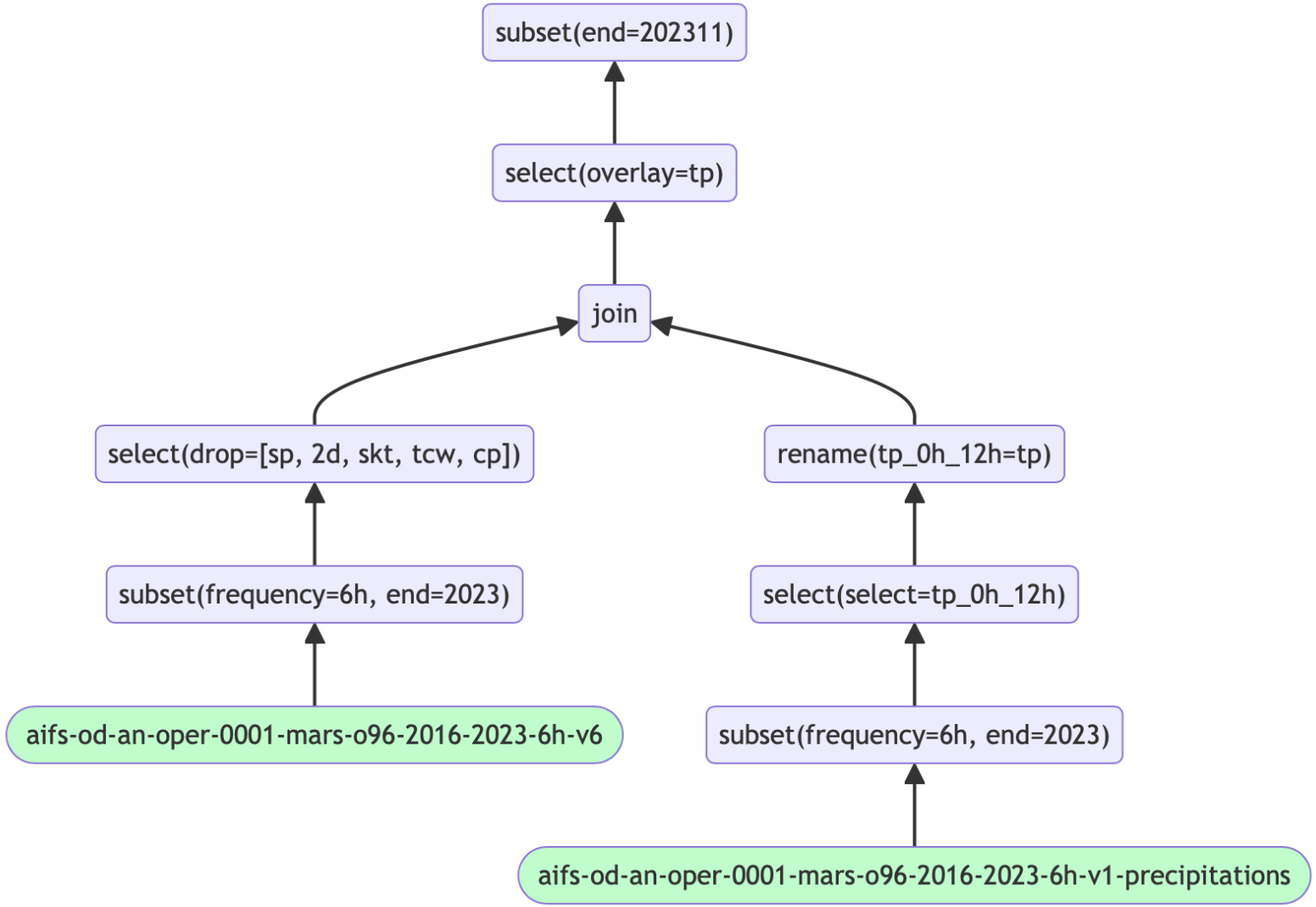
When combining datasets, the statistics of the first encountered dataset is used

Examples

```python
from anemoi.datasets import open_dataset

ds = open_dataset(
    [
        {
            "dataset": "aifs-od-an-oper-0001-mars-o96-2016-2023-6h-v6",
            "drop": ["sp", "2d", "skt", "tcw", "cp"],
            "end": 2023,
            "frequency": "6h",
        },
        {
            "dataset": "aifs-od-an-oper-0001-mars-o96-2016-2023-6h-v1-precipitations",
            "end": 2023,
            "frequency": "6h",
            "rename": {"tp_0h_12h": "tp"},
            "select": ["tp_0h_12h"],
        },
    ],
    end=202311,
)
```

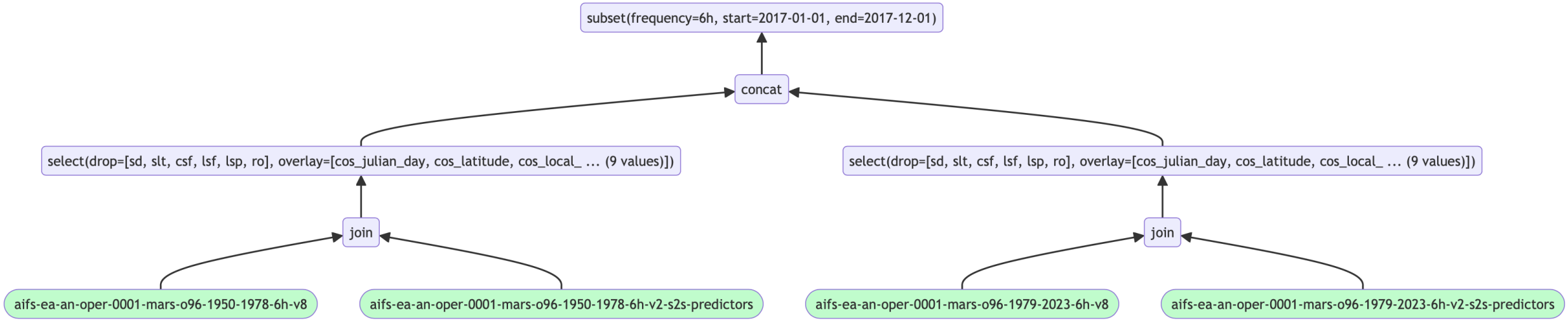


```python
from anemoi.datasets import open_dataset

ds = open_dataset(
    [
        {
            "dataset": "aifs-od-an-oper-0001-mars-n320-2016-2023-6h-v8",
            "end": 2022,
            "frequency": "6h",
        },
        {
            "dataset": "aifs-od-an-oper-0001-mars-n320-2016-2023-6h-v1-land",
            "drop": [
                "anor",
                "isor",
                "lsp",
                "stl3",
                "swvl3",
                "tsn",
                "rsn",
                "sd",
                "tvh",
                "tvl",
                "cvh",
                "cvl",
                "cl",
                "slt",
                "lai_lv",
                "lai_hv",
            ],
            "end": 2022,
            "frequency": "6h",
        },
    ],
    end=2022,
)
```

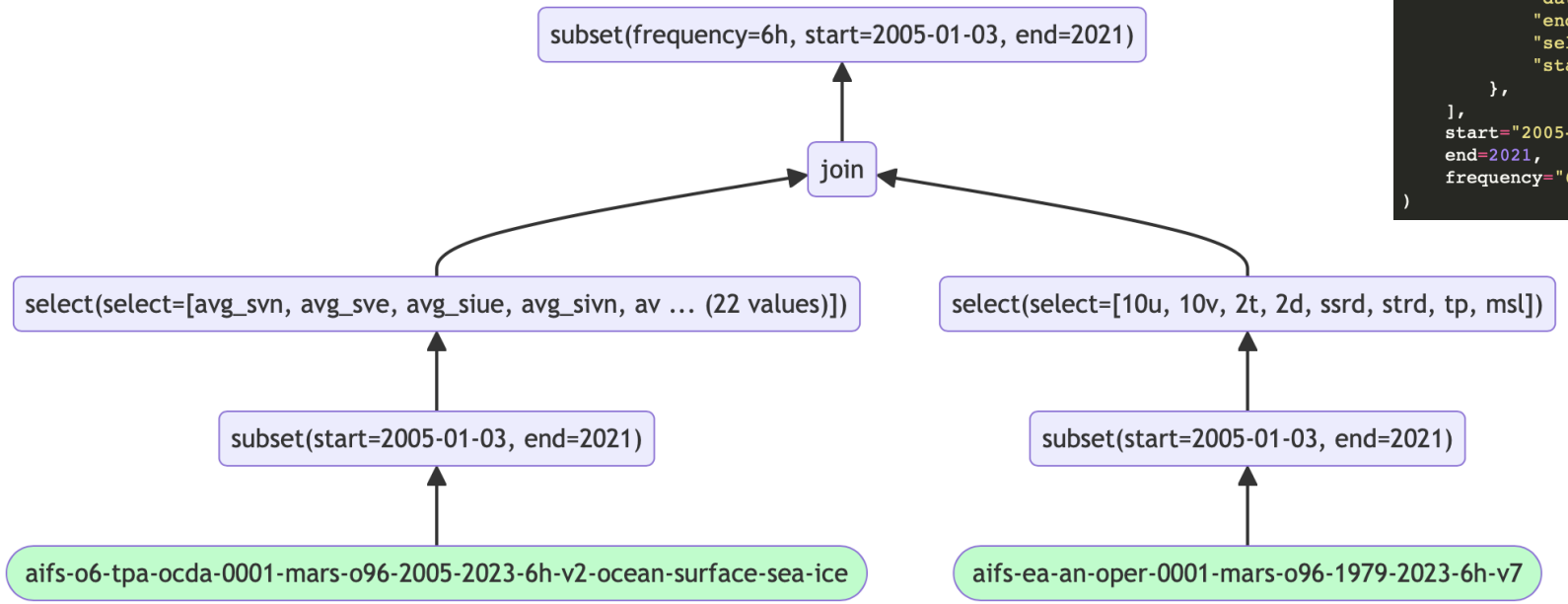


```python
from anemoi.datasets import open_dataset

ds = open_dataset(
    [
        {
            "dataset": {
                "drop": ["sd", "slt", "csf", "lsf", "lsp", "ro"],
                "join": [
                    {"dataset": "aifs-ea-an-oper-0001-mars-o96-1950-1978-6h-v8"},
                    {
                        "dataset": "aifs-ea-an-oper-0001-mars-o96-1950-1978-6h-v2-s2s-predictors"
                    },
                ],
            }
        },
        {
            "dataset": {
                "drop": ["sd", "slt", "csf", "lsf", "lsp", "ro"],
                "join": [
                    {"dataset": "aifs-ea-an-oper-0001-mars-o96-1979-2023-6h-v8"},
                    {
                        "dataset": "aifs-ea-an-oper-0001-mars-o96-1979-2023-6h-v2-s2s-predictors"
                    },
                ],
            }
        },
    ],
    frequency="6h",
    start="2017-01-01",
    end="2017-12-01",
)
```

Details

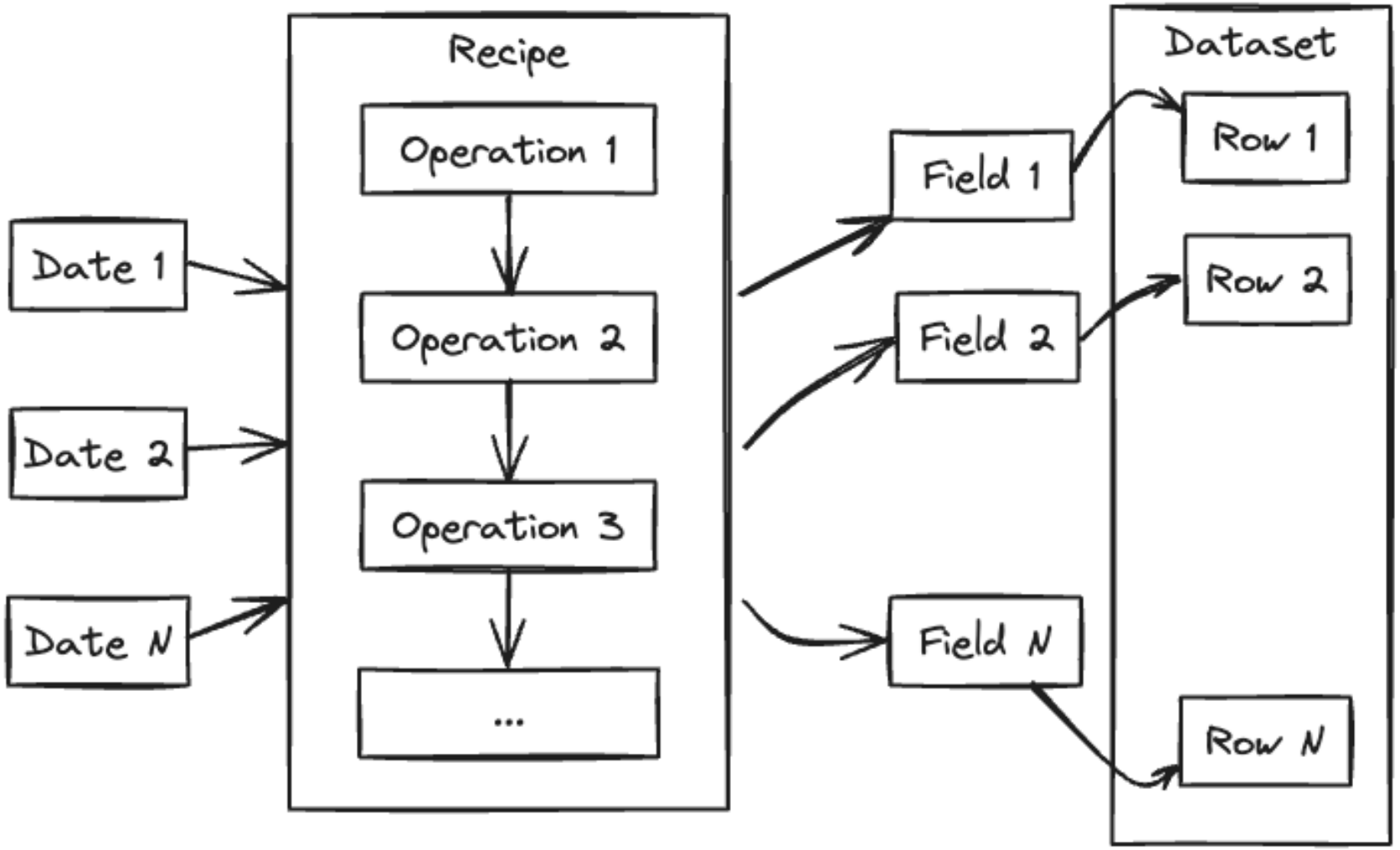


```python
from anemoi.datasets import open_dataset

ds = open_dataset(
    [
        {
            "dataset": "aifs-o6-tpa-ocda-0001-mars-o96-2005-2023-6h-v2-ocean-surface-sea-ice",
            "end": 2021,
            "select": [
                "avg_svn",
                "avg_sve",
                "avg_siue",
                "avg_sivn",
                "avg_sivol",
                "avg_snvol",
                "avg_siconc",
                "avg_icesalt",
                "avg_sialb",
                "avg_tos",
                "avg_sos",
                "avg_zos",
                "cos_latitude",
                "sin_latitude",
                "cos_longitude",
                "sin_longitude",
                "cos_solar_zenith_angle",
                "cos_julian_day",
                "cos_local_time",
                "sin_julian_day",
                "sin_local_time",
                "lsm",
            ],
            "start": "2005-01-03",
        },
        {
            "dataset": "aifs-ea-an-oper-0001-mars-o96-1979-2023-6h-v7",
            "end": 2021,
            "select": ["10u", "10v", "2t", "2d", "ssrd", "strd", "tp", "msl"],
            "start": "2005-01-03",
        },
    ],
    start="2005-01-03",
    end=2021,
    frequency="6h",
)
```



Building datasets

Datasets are built from a recipe



Recipes are YAML files

```yaml
dates:
    start: 2024-01-01T00:00:00Z
    end: 2024-01-01T18:00:00Z
    frequency: 6h

input:
    grib:
        param: [2t, msl, 10u, 10v, lsm]
        path: /path/to/some/data.grib
```

Then:

```
% anemoi-datasets create recipe.yaml dataset.zarr
```

The dates block

```
dates:
   start: 2010-01-01 00:00:00
   end: 2017-01-31 18:00:00
   frequency: 6h
   group_by: monthly
```

Note: dates a “valid dates”, not “reference dates”

The output block

```
output:
   dtype: float32
   chunking:
      dates: 1
      ensembles: 1
```

Naming variables

- Variable names are just strings without semantics
- The names are used throughout anemoi (datasets, training, inference)
- Variables are named as `"{param}_{level}"` by default.
- You can change the naming method

```
build:
  variable_naming: param
```

(Note: “param” and “variable”, as well as “level” and “levelist” are mostly synonymous when dealing with GRIB and NetCDF)

Operations

- Operations are blocks of YAML code that translates a list of dates into fields.
- Simplest operation is just a source of data

```yaml
input:
  grib:
    path: /path/to/file.grib
    param: [msl,2t]
```

Join

The join is the process of combining several sources data. Each source is expected to provide different variables at the same dates.

```
input:
  join:
    - source1:
        ...
    - source2:
        ...
    - source3:
        ...
    - ...
```

Pipe

The pipe is the process of transforming fields using filters. The first step of a pipe is typically a source. The following steps are filters.

```
input:
  pipe:
    - source
        ...
    - filter1
        ...
    - filter2
        ...
    - ...
```

Concat

The concatenation is the process of combining different sets of operation that handle different dates.

```
input:
  concat:
  - dates:
      start: 2020-12-30 00:00:00
      end: 2021-01-01 12:00:00
      frequency: 12h

    source1:
      ...

  - dates:
      start: 2021-01-02 00:00:00
      end: 2021-01-03 12:00:00
      frequency: 12h

    source2:
      ...
```

Statistics gathering

- By defaults, the statistics are not computed on the whole dataset, but on a subset of dates.
- The dates subset used to compute the statistics is defined using the following algorithm:
  - If the dataset covers 20 years or more, the last 3 years are excluded.
  - If the dataset covers 10 years or more, the last year is excluded.
  - Otherwise, 80% of the dataset is used.
  - Using user provided dates

```
statistics:
    end: 2020
```

- By default, statistics will fail if a NaN is found. You can change that:

```
statistics:
    allow_nans: true
```

- Statistics of increments can also be computed:
  - 1h, 3h, 6h 12h and 24h increments

Handling missing dates

By default, the package will raise an error if there are missing dates.

Missing dates can be handled by specifying a list of dates in the configuration file. The missing dates will be filled `np.nan` values.

```
dates:
  start: 2017-01-01 00:00:00
  end: 2017-01-31 23:00:00
  frequency: 1h

  missing:
  - 2017-01-02 00:00:00
  - 2017-01-03 00:00:00
```

Anemoi will ignore the missing dates when computing the statistics.

You can retrieve the list indices corresponding to the missing dates.

```
print(ds.missing)
```

If you access a missing date, the dataset will throw a `MissingDateError`.

- Sources

General purposes sources

Name	Description

grib	Access grib files on disk
	
	`input:`
	`  grib:`
	`  param:[msl, 2t]`
	`  path:/path/to/file.grib`

netcdf	Access NetCDF files on disk (xarray-based)
	
	`input:`
	`  netcdf:`
	`  path:/path/to/file.nc`

opendap	Access OpenDAP servers (xarray-based)
	
	`input:`
	`  opendap:`
	`  url: http://`

xarray-zarr	Access file-based or cloud-based Zarr (xarray-based)
	`input:`
	`  xarray-zarr:`
	`  url: http://`

xarray-kerchunk	Experimental: Access kerchunk datasets (xarray-based)
zenodo	Experimental: access a dataset given a Zenodo DOI

File/URL based sources (grib, NetCDF, OpenDAP, …)

Unix wildcards

```
input:
  grib:
    path: /path/to/*.grib
```

Parameter substitutions

```
input:
  grib:
    path: /path/to/data-{param}.grib
    param: [u, v]
```

Date components

```
input:
  grib:
    path: /path/to/{date:strftime(%Y%m)/{date:strftime(%Y%m%d%H)}.grib
```

Xarray-based sources

- Xarray-based sources (netcdf, opendap, zarr, etc) will expect the source dataset to follow the CF-convention

```
input:
  netcdf:

      # Selection of the fields

      variable: [temperature, geopotential]
      level: [1000, 500]

      # Control

      options: ... # kwargs passed to xarray.open_zarr or open_dataset

      patch: ... # Dictionary to update metadata

      flavour: ... # Dictionary to help parsing the metadata
```

ECMWF specific sources

Name	Descriptions

mars	Retrieve data from the MARS archive
	
	`input:`
	`  mars:`
	`  param:[msl, 2t]`
	`  levtype: sfc`

accumulations	Retrieve accumulated fields from the MARS archive (e.g. total precipitations) and compute their values over different accumulations period)
	
	`input:`
	`  accumulations:`
	`  accumulation_period: 6`
	`  class: ea`
	`  param: [tp, cp, sf]`
	`  levtype: sfc`

hindcasts	Retrieve reforecasts from the MARS archive

Specialised sources (rely on other sources)

Name	Description
recentre	Recentre an ensemble source on another. Used to recentre the ensemble around the operational analysis. This source uses two other sources: one for the ensemble, one for the new centre.
repeated-dates	Use to repeat a constant field or a climatological field at each dates of a dataset (more in follow up slide)
forcings	A collection of fields that can be computed on the fly from the date or the grid (more in follow up slide)

forcings

The purpose of forcings is to provide fields with values that only depend on the grid cell and/or the time.

Because the source needs to generate fields on the same grids as the other, it requires a template field. This is provided in the recipe with the `template` keyword:

```
input:
  join:
    - source1:
      args1: value1
      args2: value2
    - forcings:
      template: ${input.join.0.source1}
      param:
        - insolation
        - cos_julian_day
        - sin_julian_day
```

The value `${input.join.0.source1}` is the “path” to the first source, starting from the root of the recipe.

forcings

Name	Description	Periodic	Range
latitude	Each grid point has the value of its latitude in degrees.	No	$[-90,90]$
cos_latitude	Each grid point has the value of $\cos(\text{latitude}/180 * \pi)$	Yes	$[-1,1]$
sin_latitude	Each grid point has the value of $\sin(\text{latitude}/180 * \pi)$	Yes	$[-1,1]$
longitude	Each grid point has the value of its longitude in degrees.	No	$[-180,360)$
cos_longitude	Each grid point has the value of $\cos(\text{longitude}/180 * \pi)$.	Yes	$[-1,1]$
sin_longitude	Each grid point has the value of $\sin(\text{longitude}/180 * \pi)$.	Yes	$[-1,1]$
julian_day	The Julian day is al number of days since the 1st or January at 00:00 of the current year.	No	$[0,365)$ or $[0,366)$
cos_julian_day	Each grid point has the value of $\cos(\text{julian\_day}/365.25 * 2 * \pi)$.	Yes	$[-1,1]$
sin_julian_day	Each grid point has the value of $\sin(\text{julian\_day}/365.25 * 2 * \pi)$.	Yes	$[-1,1]$
local_time	Each grid point has the value of the local time in hours (no time zone information is used).	No	$[0,24)$
cos_local_time	Each grid point has the value of $\cos(\text{local\_time}/24 * 2 * \pi)$.	Yes	$[-1,1]$
sin_local_time	Each grid point has the value of $\sin(\text{local\_time}/24 * 2 * \pi)$.	Yes	$[-1,1]$
insolation	This is an alias for the cos_solar_zenith_angle field.	Yes	$[0,0.325]$

repeated-dates

constant

```yaml
repeated-dates:
  mode: constant
  source:
      xarray-zarr:
        url: dem.zarr
        variable: dem
```

climatology

```yaml
repeated-dates:
    mode: climatology
    year: 2019
    day: 15
    source:
        grib:
          path: some/path/to/data.grib
          param: [some_param]
```

closest

```yaml
repeated-dates:
    mode: closest
    frequency: 24h
    maximum: 30d
    skip_all_nans: true
    source:
      grib:
          path: path/to/data.grib
          param: [some_param]
```

Recenter

```yaml
data_sources:
  members_source:
    mars:
      class: ea
      expver: "0001"
      grid: 20.0/20.0
      levtype: sfc
      param: [10u, 10v, 2t]
      type: an
      stream: enda
      number: [1, 2, 3, 4, 5, 6, 7, 8, 9]

  center_source:
    mars:
      class: ea
      expver: "0001"
      grid: 20.0/20.0
      levtype: sfc
      param: [10u, 10v, 2t]
      type: an
      stream: oper

input:
  recentre:
    centre: ${data_sources.center_source}
    members: ${data_sources.members_source}
```

- Filters

Filters

Name	Description
apply-mask	Apply a mask to the fields (set values to NaN according to a mask)
rotate_wind/unrotate_wind	Rotate wind vectors
rename	Rename variables
regrid	Interpolate input fields to a different grid
rescale	Apply unit conversions
orog_to_z	Convert orography (m) to z (geopotential height)
uv-2-ddff/ddff-2-uv	Convert wind components between cartesian and polar coordinates
wz_to_w	Convert geometric vertical velocity (m/s) to vertical velocity (Pa/s)

- This is only a small selection, work in progress
- Some sites implement their own filters

rename

```
input:
  pipe:
    - netcdf:
       path: myfile.nc

    - rename:
        param:
          temperature_2m: 2t
          temperature_850hPa: t_850
```

Creating a dataset incrementally

Building the dataset

```
anemoi-datasets init dataset.yaml dataset.zarr --overwrite
```

You can then load the dataset in parts with the load command in parallel.

```
anemoi-datasets load dataset.zarr --part 1/20
anemoi-datasets load dataset.zarr --part 2/20
...
anemoi-datasets load dataset.zarr --part 20/20
```

Once you have loaded all the parts, finalise the dataset with the finalise command.

```
anemoi-datasets finalise dataset.zarr
```

You can follow the progress of the dataset creation with the inspect command.

```
anemoi-datasets inspect dataset.zarr
```

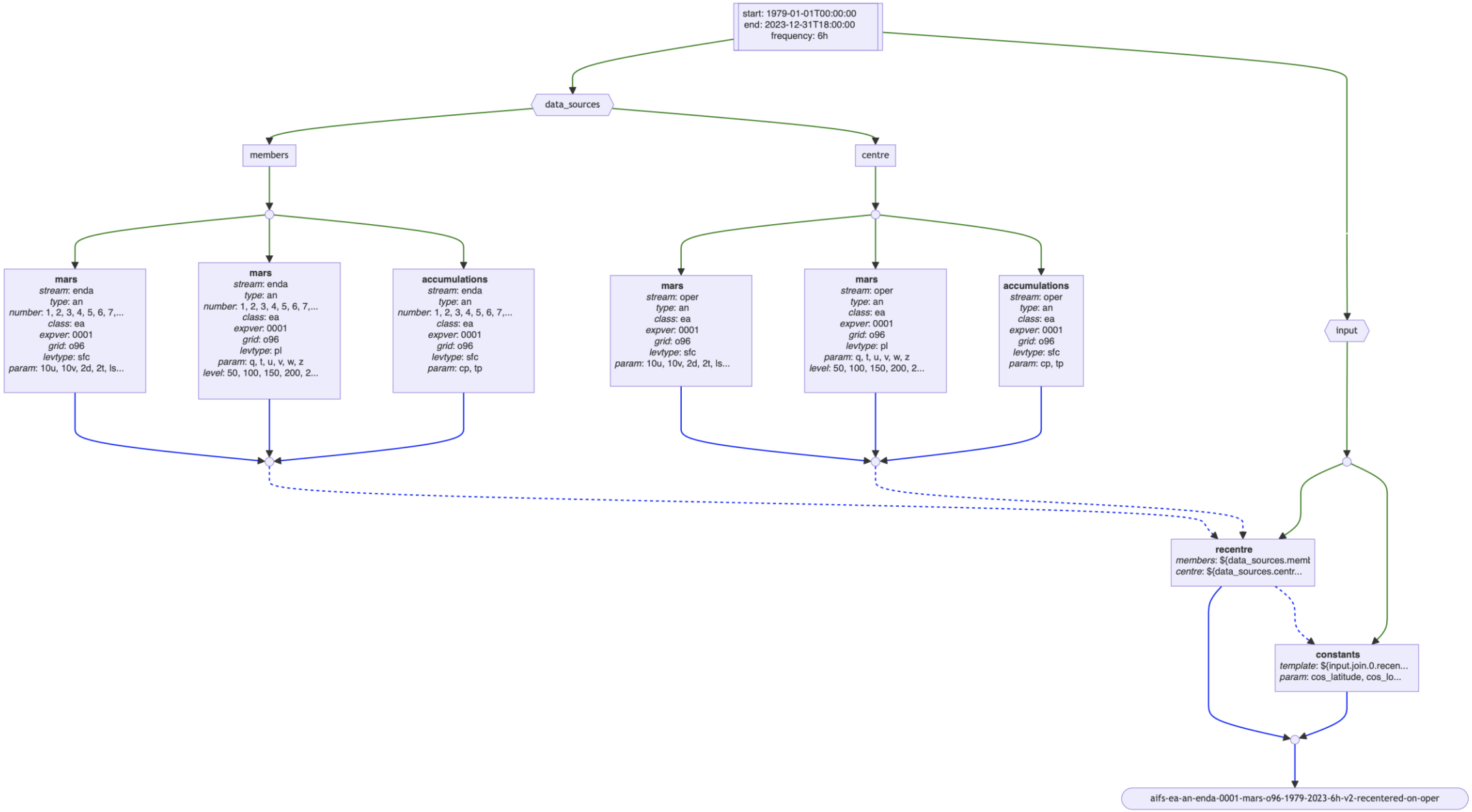
It is possible that some temporary files are left behind at the end of the process.

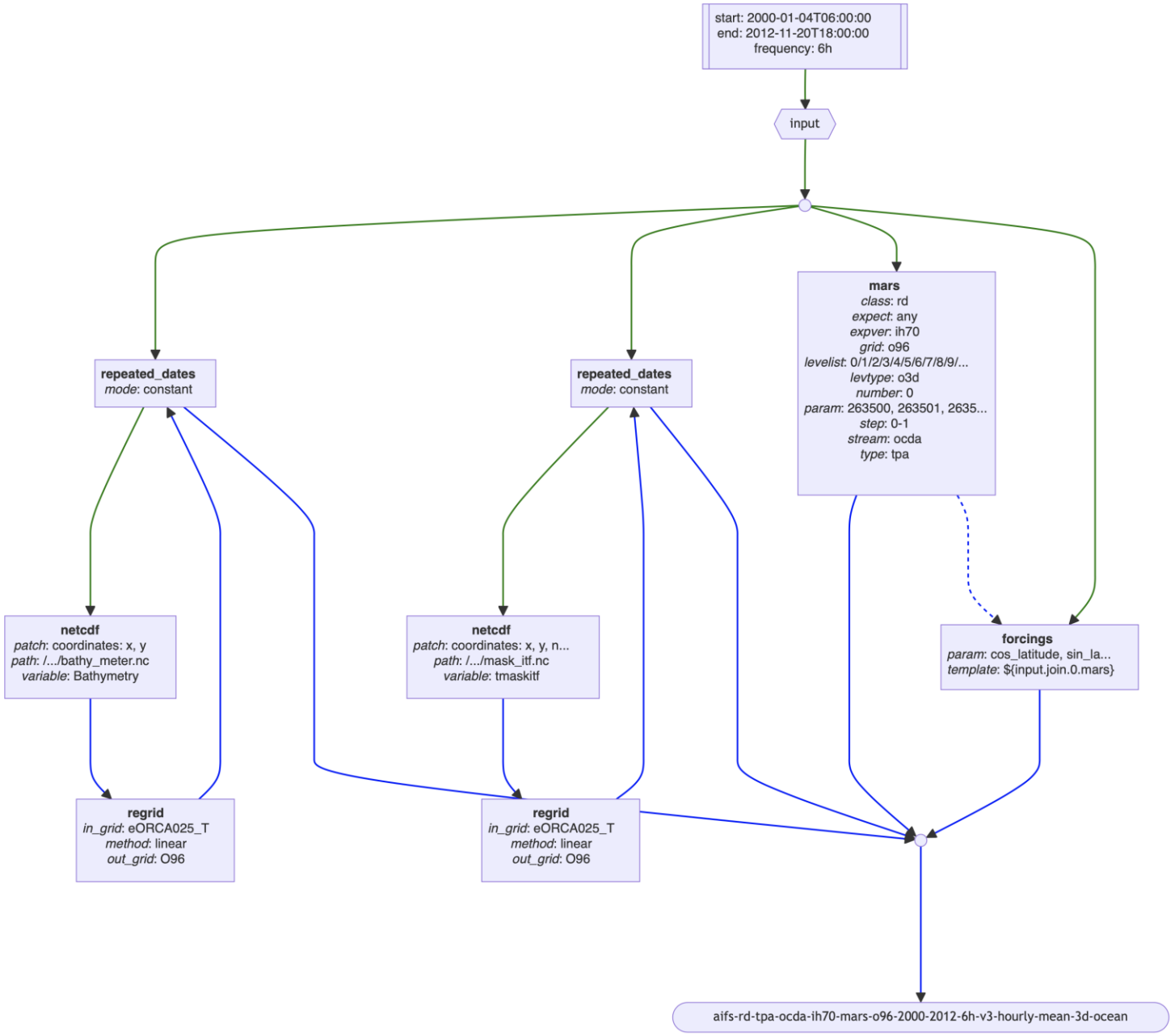
```
anemoi-datasets cleanup dataset.zarr
```

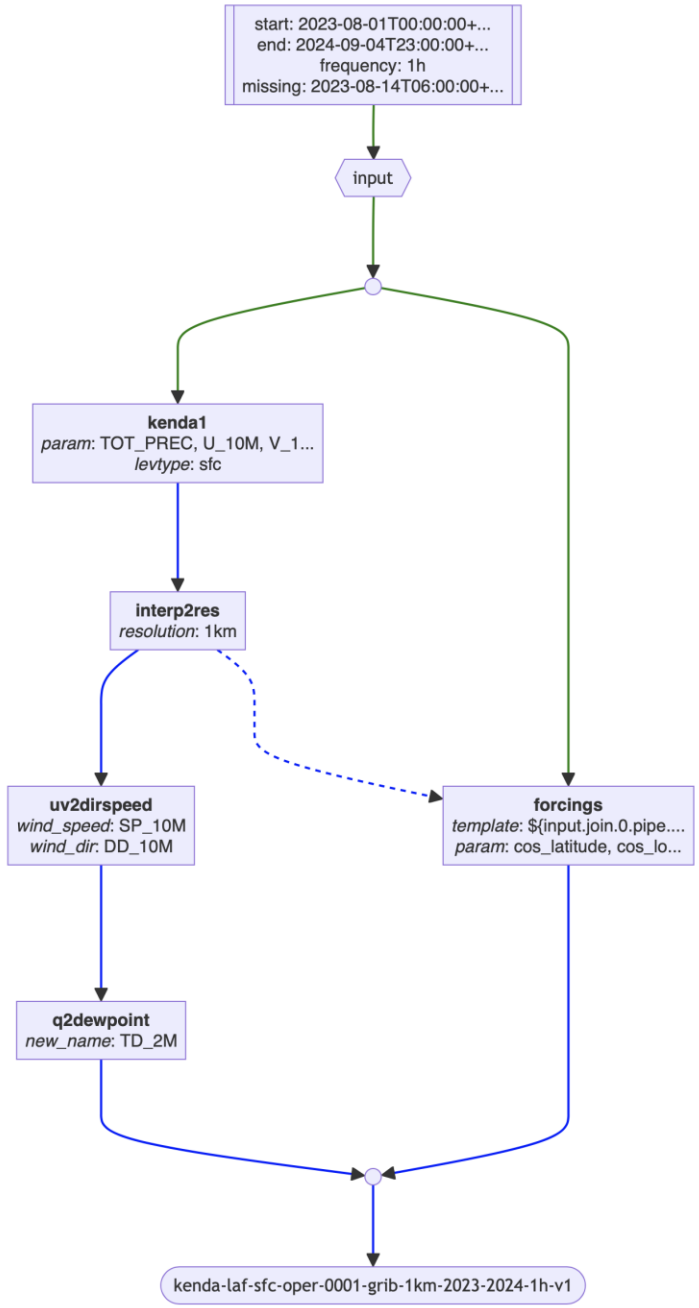
Extending

- Adding new sources or new filters
  - Can be done by contributing to anemoi-datasets or anemoi-transform
  - A plugin system is also under construction
- Sources or filters that are of general interest should be contributed to the main code base

Examples


start: 1979-01-01T00:00:00
end: 2023-12-31T18:00:00
frequency: 6h
data_sources
members
centre
mars
stream: enda
type: an
number: 1, 2, 3, 4, 5, 6, 7,...
class: ea
expver: 0001
grid: o96
levtype: sfc
param: 10u, 10v, 2d, 2t, ls...
mars
stream: enda
type: an
number: 1, 2, 3, 4, 5, 6, 7,...
class: ea
expver: 0001
grid: o96
levtype: pl
param: q, t, u, v, w, z
level: 50, 100, 150, 200, 2...
accumulations
stream: enda
type: an
number: 1, 2, 3, 4, 5, 6, 7,...
class: ea
expver: 0001
grid: o96
levtype: sfc
param: cp, tp
mars
stream: oper
type: an
class: ea
expver: 0001
grid: o96
levtype: sfc
param: 10u, 10v, 2d, 2t, ls...
mars
stream: oper
type: an
class: ea
expver: 0001
grid: o96
levtype: pl
param: q, t, u, v, w, z
level: 50, 100, 150, 200, 2...
accumulations
stream: oper
type: an
class: ea
expver: 0001
grid: o96
levtype: sfc
param: cp, tp
input
recentre
members: ${data_sources.memt
centre: ${data_sources.centr...
constants
template: ${input.join.0.recen...
param: cos_latitude, cos_lo...
aifs-ea-an-enda-0001-mars-o96-1979-2023-6h-v2-recentered-on-oper


start: 2000-01-04T06:00:00
end: 2012-11-20T18:00:00
frequency: 6h
input
repeated_dates
mode: constant
repeated_dates
mode: constant
mars
class: rd
expect: any
expver: ih70
grid: o96
levelist: 0/1/2/3/4/5/6/7/8/9/...
levtype: o3d
number: 0
param: 263500, 263501, 2635...
step: 0-1
stream: ocda
type: tpa
netcdf
patch: coordinates: x, y
path: /.../bathy_meter.nc
variable: Bathymetry
netcdf
patch: coordinates: x, y, n...
path: /.../mask_itf.nc
variable: tmaskitf
forcings
param: cos_latitude, sin_la...
template: ${input.join.0.mars}
regrid
in_grid: eORCA025_T
method: linear
out_grid: O96
regrid
in_grid: eORCA025_T
method: linear
out_grid: O96
aifs-rd-tpa-ocda-ih70-mars-o96-2000-2012-6h-v3-hourly-mean-3d-ocean


start: 2023-08-01T00:00:00+...
end: 2024-09-04T23:00:00+...
frequency: 1h
missing: 2023-08-14T06:00:00+...
input
kenda1
param: TOT_PREC, U_10M, V_1...
levtype: sfc
interp2res
resolution: 1km
uv2dirspeed
wind_speed: SP_10M
wind_dir: DD_10M
forcings
template: ${input.join.0.pipe....
param: cos_latitude, cos_lo...
q2dewpoint
new_name: TD_2M
kenda-laf-sfc-oper-0001-grib-1km-2023-2024-1h-v1

Questions?