DISCOVER ANEMOI · WEBINAR SERIES · PART 3

Anemoi Training + Inference

Creating Machine Learning weather models, and how to run them.

PRESENTER
Ana Prieto Nemesio
Machine Learning Team Lead
ana.prietonemesio@ecmwf.int

PRESENTER
Harrison Cook
Research Software Engineer
harrison.cook@ecmwf.int

DISCOVER ANEMOI · WEBINAR SERIES · PART 3

Anemoi Training + Inference

Q&A – dedicated session after each presentation webinar, via the chat

Recordings – all recordings will be made available on the event webpage

No AI meeting assistants – please don't add them for this webinar

Contact – training@ecmwf.int

PRESENTER
Ana Prieto Nemesio
Machine Learning Team Lead
ana.prietonemesio@ecmwf.int

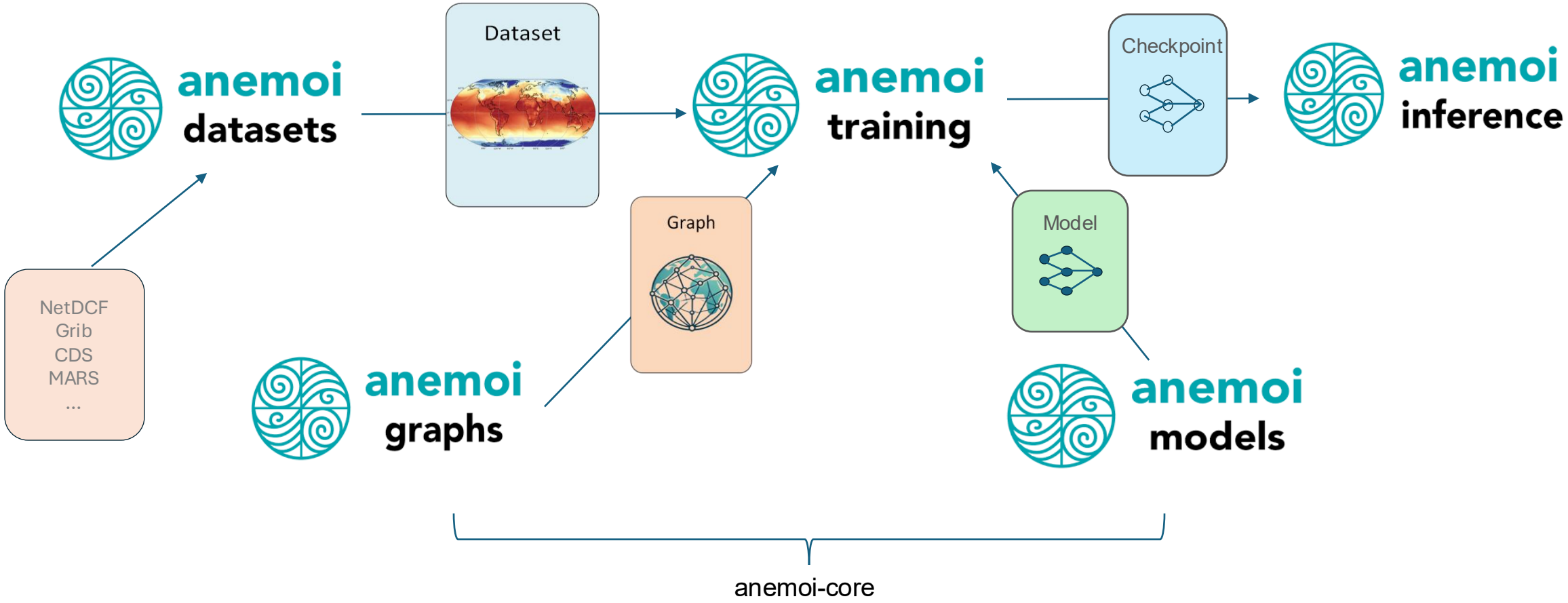
PRESENTER
Harrison Cook
Research Software Engineer
harrison.cook@ecmwf.int

Anemoi Training

Ana Prieto Nemesio

An overview of the main components

Anemoi is a family of packages. Each owns one stage of the journey from raw archives to a running forecast.



ANEMOI-TRAINING

A configurable training pipeline

WHAT IT GIVES YOU

- Built on **PyTorch Lightning** and **Hydra** — modify key components without code changes
- Multi-node / multi-GPU support out of the box
- Deterministic *and* probabilistic training
- Callbacks for profiling, evaluating, plotting and logging
- Experiment tracking with e.g. **MLflow**

INTERFACES WITH

`anemoi-datasets`
via data loaders

`anemoi-models`
via Hydra configuration

`anemoi-inference`
via metadata-rich checkpoints

Models you can train

Forecasting

`Global` whole-Earth deterministic forecast

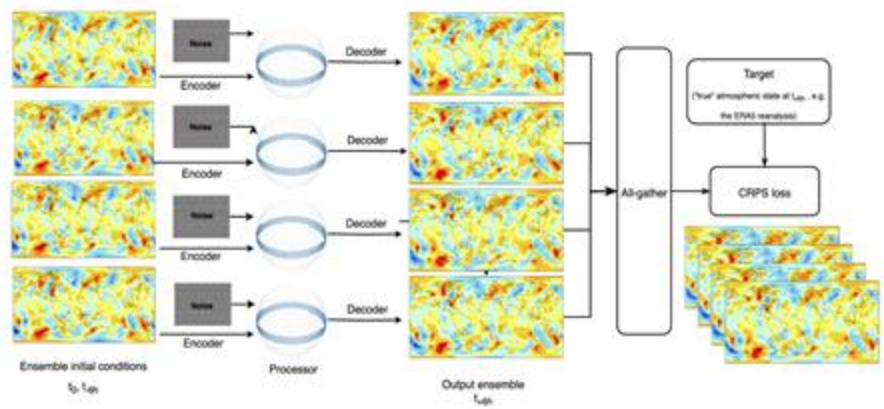
`LAM` limited-area model

`Stretched grid` global with a high-res region

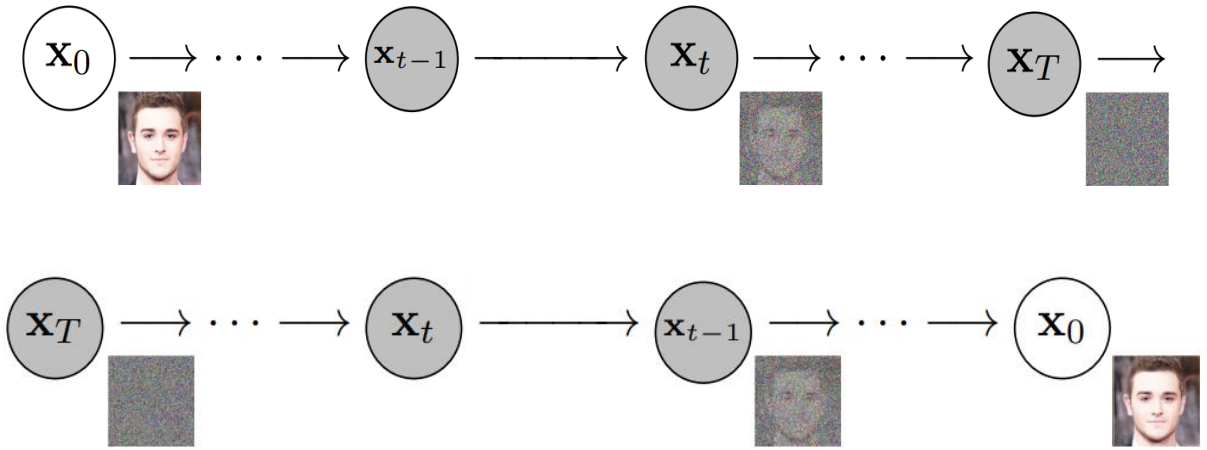
Probabilistic

`CRPS-based` ensembles trained on a probabilistic score

`Diffusion` generative model · Lang et al. 2024b



Lang et al. 2024b



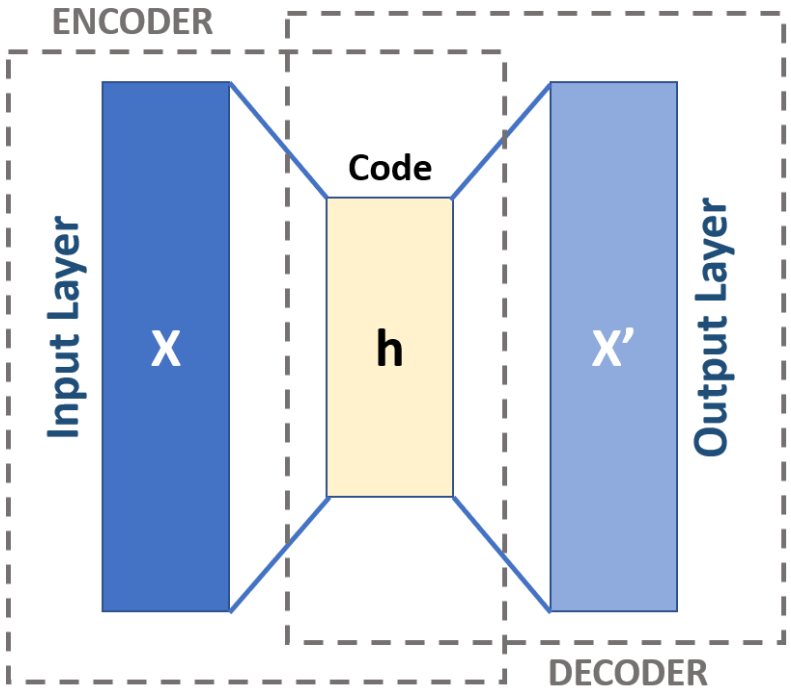
MODELS YOU CAN TRAIN

Autoencoder

LATENT-SPACE MODEL

`field` → **`latent`** → `field`

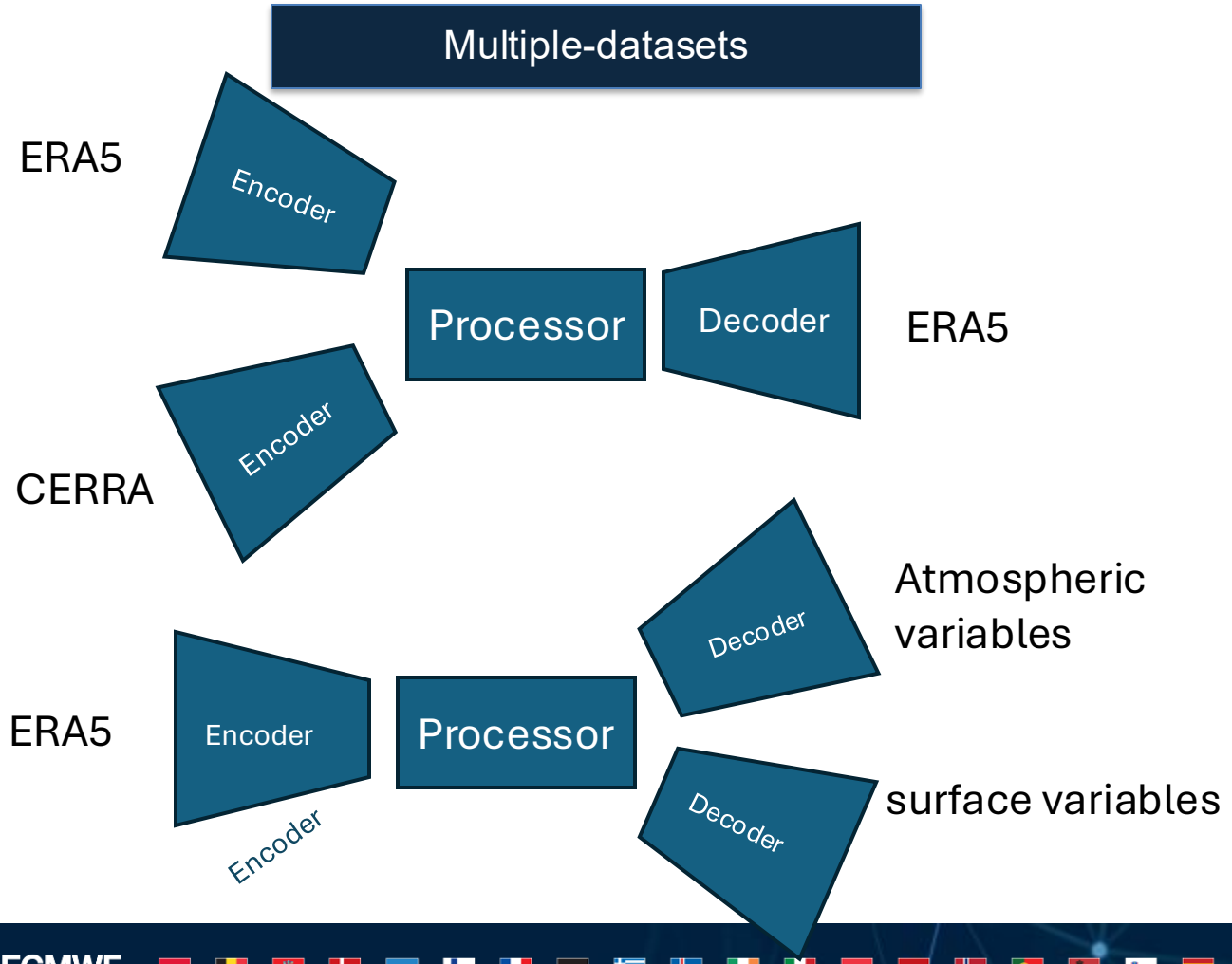
Compress the atmosphere into a learned latent space, then reconstruct it.

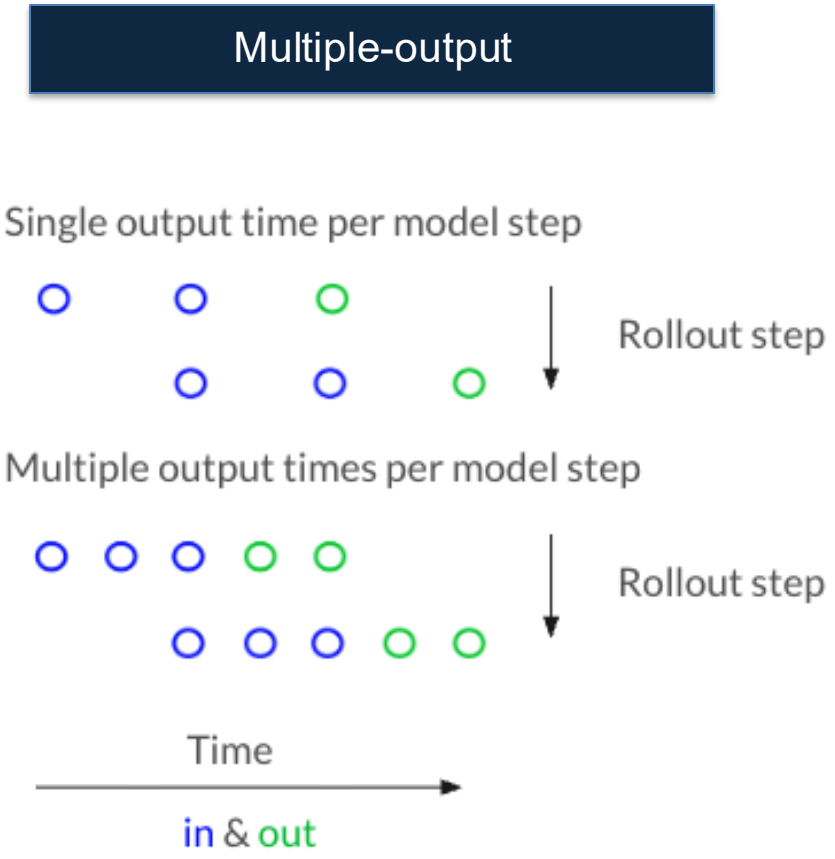


- **Enable scalable latent-space modelling**
- **Support superresolution & reconstruction tasks**
- **Learn compressed spatial representations**

Multiple datasets, multiple outputs

One encoder per input source, one decoder per output stream — around a shared processor.



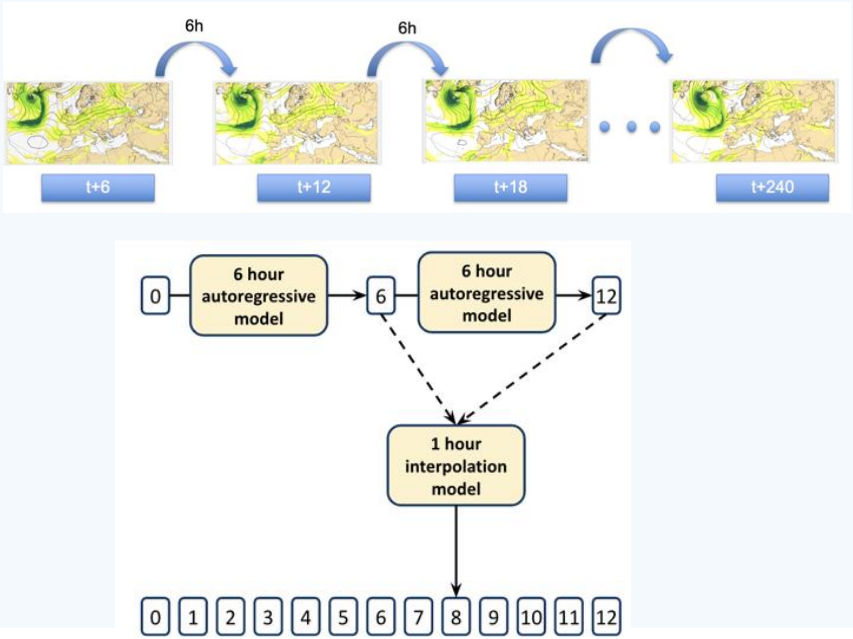


MODELS YOU CAN TRAIN

Downscaling

Temporal downscaling

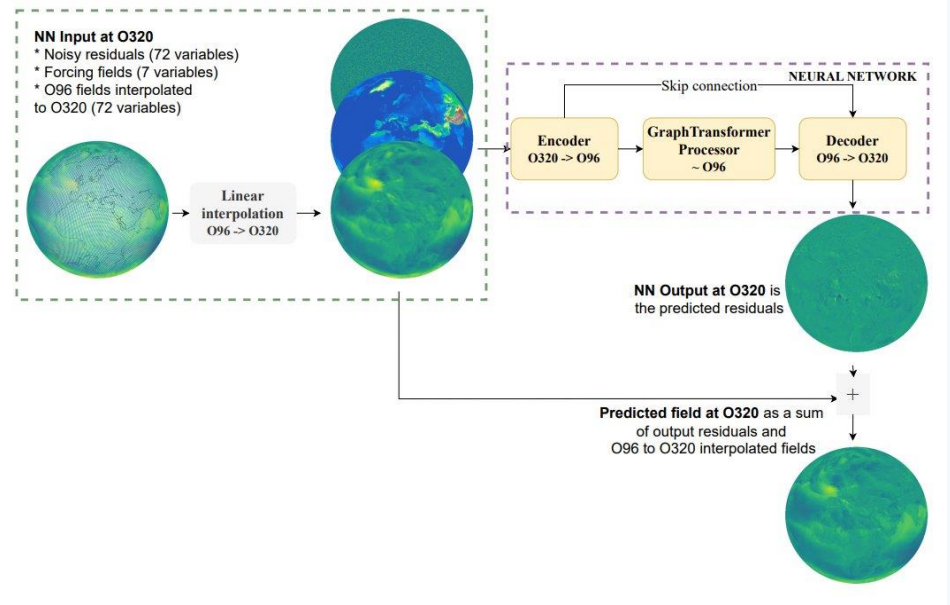
Increase the time resolution — e.g. a 1-hour interpolation model filling between 6-hourly steps..



Spatial downscaling

COMING SOON

Increase the spatial resolution — map a coarse field to a finer grid.



Getting started with anemoi-training

Configuration, the command line, and switching components without touching code.

GETTING STARTED

Interfaces overview

How the Trainer pulls the sibling packages together at runtime.

`BaseAnemoiReader` · `open_dataset`

opens your **anemoi-datasets**

`Model`

anemoi-models, built from Hydra config

`GraphCreator`

anemoi-graphs, built from Hydra config

→

IN THE

Trainer

assembles data, model & graph, then trains

→

PRODUCES

Checkpoint object

consumed by anemoi-inference

CONFIGURATION

Training is configuration-first

Set up to enable modifying key components of the model and training **without code changes**.

Hydra

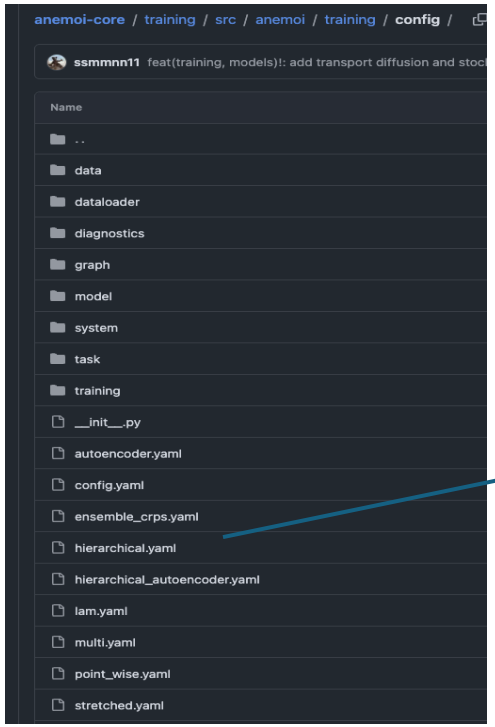
The config system underneath — compose, override and reproduce runs.

Templates

Ready-made starting points for common use cases.

YAML hierarchy

Layered files — defaults at the bottom, your overrides on top.




```yaml
defaults:
- data: zarr
- dataloader: native_grid
- diagnostics: evaluation
- system: example
- graph: multi_scale
- model: gnn
- task: forecaster
- training: single
- _self_


# set to true to switch on config validation
config_validation: True
```

```yaml
num_channels: 512
cpu_offload: False

keep_batch_sharded: True

model:
  _target_: anemoi.models.models.AnemoiModelEncProcDec
  hidden_nodes_name: "hidden"
  latent_skip: True

layer_kernels:
  # The layer_kernels can be adjusted per model component, but are defined here for convenience.
  LayerNorm:
    # The GNN requires the autocast layer norm, otherwise its memory usage is too high.
    _target_: anemoi.models.layers.normalization.AutocastLayerNorm
  Linear:
    _target_: torch.nn.Linear
  Activation:
    _target_: torch.nn.GELU

processor:
  _target_: anemoi.models.layers.processor.GNNProcessor
  trainable_size: ${model.trainable_parameters.hidden2hidden}
  sub_graph_edge_attributes: ${model.attributes.edges}
  num_layers: 16
  num_chunks: 2
  mlp_extra_layers: 0
  mlp_hidden_ratio: 1.0 # GNN only; use 1.0 for no expansion, ~2.67 for gated variants (glu/swiglu
  mlp_implementation: mlp
  cpu_offload: ${model.cpu_offload}
  gradient_checkpointing: True
  layer_kernels: ${model.layer_kernels}
```

CONFIGURATION

Eight config groups

`data`	These fields define the variables used by the model and how to normalise them
`diagnostics`	Diagnostics produced during training with the validation dataset.
`graph`	Your graph recipe
`task`	defines the temporal I/O structure of a training sample: which time steps are loaded as model inputs and which are used as prediction targets.
`dataloader`	These fields define parameters about we load the data specified in "data/".
`system`	HPC / device config, plus input & output paths
`model`	The graph-neural-network architecture
`training`	Hyperparameters — learning rate, loss function, …

COMMAND LINE

Launching a run

GENERATE A USER CONFIG FOLDER

```
>>> anemoi-training config generate
```

TRAIN WITH DEFAULTS

```
>>> anemoi-training train
```

TRAIN WITH YOUR SETTINGS

```
>>> anemoi-training train --config-name=my_config
```

EXPECT A CRASH

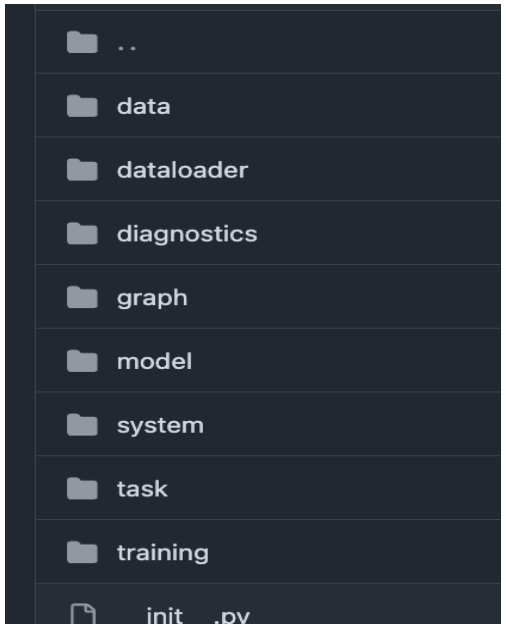
The first run intentionally stops — it doesn't know where your data is.

Placeholders are marked `???`.

THREE WAYS TO SET VALUES

1. Edit the copied config folder directly
2. Add overrides in the top-level config
3. Pass CLI overrides

How to point to where your data is stored




```yaml
dataset: ???
graph: ???
# To load the graph from a fi
# If no such file exists, the
loss_matrices_path: null
truncation: null
truncation_inv: null
warm_start: null
```

Option 1 (in low-level config)

- Picked up by top-level configs
- Can edit here directly
- Sets a default for all top-level configs pointing to this file
- Difficult to port to other configurations

Option 2 (add overrides in top-level config)

```yaml
system:
  input:
    graph: graph_anemoi_new_${data.resolution}.pt
    dataset: aifs-ea-an-oper-0001-mars-${data.resolution}-1979-2022-6h-v6
    truncation: ${data.resolution}-o32-linear.mat.npz
    truncation_inv: o32-${data.resolution}-linear.mat.npz
    loss_matrices_path: null
```

Option 3 (as cli override)

```
>>> anemoi-training train --config-name=config.yaml system.input.dataset="my_dataset_name.zarr"
```

COMMAND LINE

Pointing to your data

The same three mechanisms, applied to one question: where is the dataset?

OPTION 1

In the low-level config

Edit directly. Picked up by top-level configs, but **difficult to port** elsewhere.

OPTION 2

Override in top-level config

Sets a default for every top-level config pointing to this file.

OPTION 3

As a CLI override

A one-off, fully portable. No file edits.

```
>>> anemoi-training train --config-name=config.yaml system.input.dataset="my_dataset.zarr"
```

CONFIGURATION

Swap the model in one line

Each architecture is a YAML file under `model/`. Select one at the command line.

GNN.yml

```
num_channels: 512
processor: GNNProcessor
num_layers: 16
```

```
train model=gnn
```

GraphTransformer.yml

```
num_channels: 1024
processor: GraphTransformer
num_heads: 16
```

```
train model=graphtransformer
```

Transformer.yml

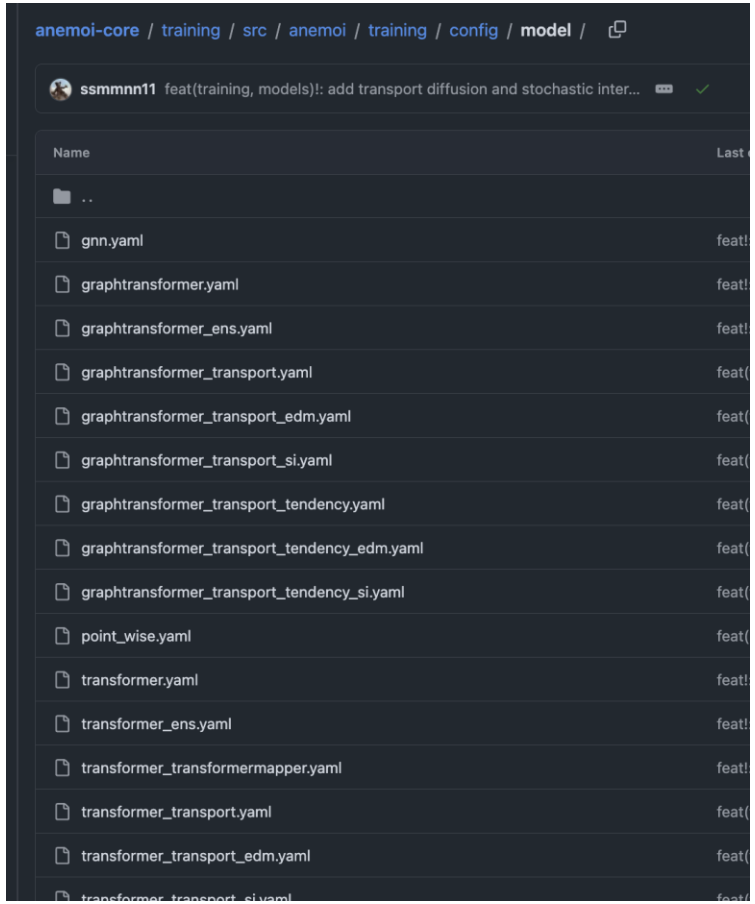
```
num_channels: 1024
processor: Transformer
window_size: 512
```

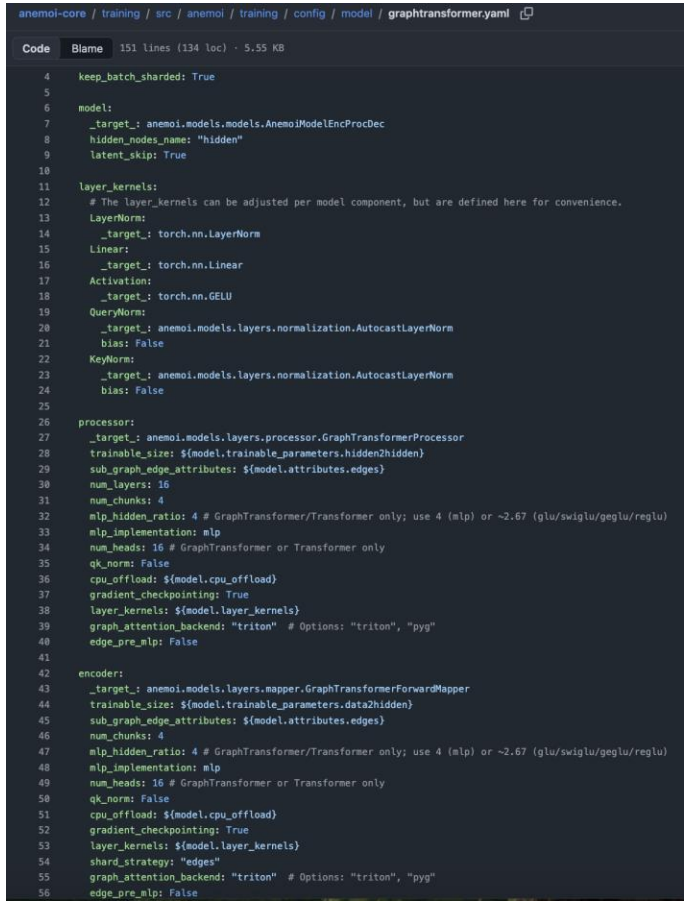
```
train model=transformer
```

Easy to switch components | Reproducible training | Easy to extend

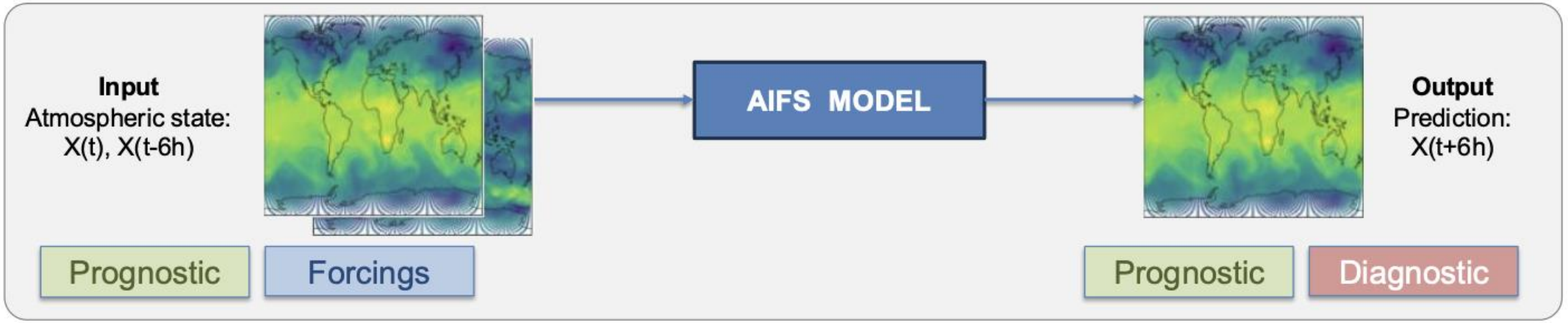
The model configs, in the repo

Real files under `config/model/` — copy one and adapt.





Classifying your variables



Prognostic

Input *and* output — variables like temperature and humidity that the model evolves.

Forcings

Input-only — land-sea mask, solar insolation. Static or dynamic. In anemoi, "forcing" simply means input-only.

Diagnostics

Output-only — precipitation and derived quantities we predict but don't feed back as inputs.

Classifying your variables

anemoi-core / training / src / anemoi / training / config / dataloader / native_grid.yaml

Code | Blame | 82 lines (73 loc) · 2.16 KB

```yaml
# runs only N training batches [N = integer | null]
# if null then we run through all the batches
limit_batches:
  training: null
  validation: null
  test: 20

# ============
# Dataloader definitions
# These follow the anemoi-datasets patterns
# You can make these as complicated for merging as you like
# See https://anemoi-datasets.readthedocs.io
# ============

# Pointers to datasets and model run info
dataset: ${system.input.dataset}
model_run_info: null  # Add for non-analysis training

training:
  datasets:
    data:
      dataset_config:
        dataset: ${dataloader.dataset}
        frequency: ${data.frequency}
        drop: []
      start: null
      end: 2020
      trajectory: ${dataloader.model_run_info}

validation:
  datasets:
    data:
      dataset_config:
        dataset: ${dataloader.dataset}
        frequency: ${data.frequency}
        drop: []
      start: 2021
      end: 2021
      trajectory: ${dataloader.model_run_info}

test:
  datasets:
    data:
      dataset_config:
        dataset: ${dataloader.dataset}
        frequency: ${data.frequency}
        drop: []
      start: 2022
      end: null
      trajectory: ${dataloader.model_run_info}
```

```yaml
format: zarr
# Time frequency requested from dataset
frequency: 6h

datasets:
  data: # user-defined name for the dataset
    # features that are not part of the forecast state
    # but are used as forcing to generate the forecast state
    forcing:
    - "cos_latitude"
    - "cos_longitude"
    - "sin_latitude"
    - "sin_longitude"
    - "cos_julian_day"
    - "cos_local_time"
    - "sin_julian_day"
    - "sin_local_time"
    - "insolation"
    - "lsm"
    - "sdor"
    - "slor"
    - "z"
    # features that are only part of the forecast state
    # but are not used as the input to the model
    diagnostic:
    - tp
    - cp

    # const_imputer:
    #   default: "none"
    #   0: []

    # processors including imputers and normalizers are applied in order of definition
    # the order of definition from this file is preserved when this config is included
    # and new processors in the main config file are added at the end of the dictionary
    processors:
      normalizer:
        _target_: anemoi.models.preprocessing.normalizer.InputNormalizer
        config:
          default: "mean-std"
```

Training settings and Tasks

anemoi-core / training / src / anemoi / training / config / training / single.yaml

Code | Blame | 120 lines (100 loc) · 3.94 KB

```yaml
# run in deterministic mode ; slows down
deterministic: False

# miscellaneous
precision: 16-mixed

# gradient accumulation across K batches, K >= 1 (if K == 1 then no accumulation)
# the effective batch size becomes num-devices * batch_size * K
accum_grad_batches: 1

num_sanity_val_steps: 6

# clipp gradients, 0 : don't clip, default algorithm: norm, alternative: value
gradient_clip:
  val: 32.
  algorithm: value

# select training method
method:
  _target_: anemoi.training.train.methods.SingleTraining

# select strategy
strategy:
  _target_: anemoi.training.distributed.strategy.DDPGroupStrategy
  num_gpus_per_model: ${system.hardware.num_gpus_per_model}
  read_group_size: ${dataloader.read_group_size}

# dynamic rescaling of the loss gradient
# see https://arxiv.org/pdf/2306.06079.pdf, section 4.3.2
# don't enable this by default until it's been tested and proven beneficial
loss_gradient_scaling: False

# Validation metrics calculation,
# This may be a list, in which case all metrics will be calculated
# and logged according to their name.
# These metrics are calculated in the output model space, and thus
# have undergone postprocessing.
validation_metrics:
  datasets:
    data: # user-defined key in data
      # loss class to initialise
      mse:
        _target_: anemoi.training.losses.MSELoss
        # Scalers to include in loss calculation
        # Cannot scale over the variable dimension due to possible remappings.
        # Available scalers include:
        # - 'loss_weights_mask': Giving imputed NaNs a zero weight in the loss function
        scalers: ["node_weights", "time_steps"]
        # other kwargs
        ignore_nans: True
```

defines the temporal I/O structure of a trainingsample: which time steps are loaded as modelinputs and which are used as prediction targets.

anemoi-core / training / src / anemoi / training / config / task / forecaster.yaml

ssmmnn11 fix(training): enable coupling of validation rollout to training roll...

Code | Blame | 15 lines (13 loc) · 381 Bytes

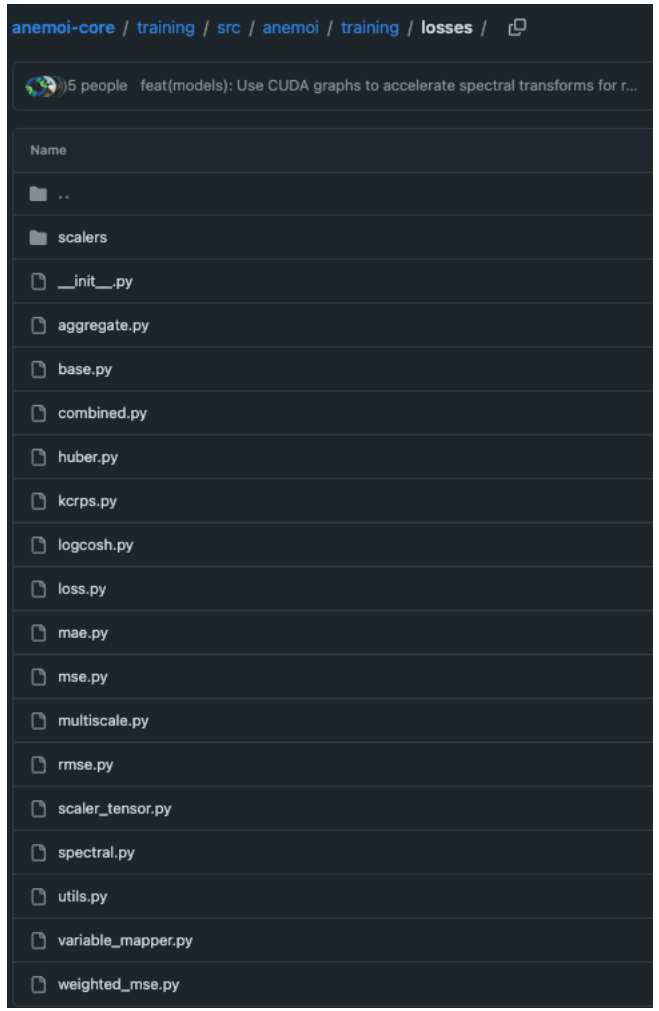
```yaml
_target_: anemoi.training.tasks.Forecaster
multistep_input: 2
multistep_output: 1
timestep: "6H"

# Rollout configuration for autoregressive forecasting
rollout:
  start: 1
  # increase rollout every n epochs
  epoch_increment: 0
  # maximum rollout to use
  maximum: 1

# number of rollout steps to use for validation, or null to use the training rollout
validation_rollout: null
```

Loss function

`anemoi/training/losses/`




```yaml
datasets:
  data:
    _target_: anemoi.training.losses.MSELoss
    # Scalers to include in loss calculation
    # A selection of available scalers are listed in training/scalers.
    # '*' is a valid entry to use all `scalers` given, if a scaler is to be excluded
    # add `!scaler_name`, i.e. ['*', '!scaler_1'], and `scaler_1` will not be added.
    scalers: ['pressure_level', 'general_variable', 'node_weights', 'time_steps']
    ignore_nans: False
```

Loss functions

anemoi/training/losses/

MSELoss	Mean squared error — the default workhorse
RMSELoss	Root mean squared error
MAELoss	Mean absolute error
HuberLoss	Huber loss — robust to outliers
LogCoshLoss	Log-cosh loss
More….	And a plug & play architecture


```yaml
datasets:
  data:
    _target_: anemoi.training.losses.MSELoss
    # Scalers to include in loss calculation
    # A selection of available scalers are listed in training/scalers.
    # '*' is a valid entry to use all `scalers` given, if a scaler is to be excluded
    # add `!scaler_name`, i.e. ['*', '!scaler_1'], and `scaler_1` will not be added.
    scalers: ['pressure_level', 'general_variable', 'node_weights', 'time_steps']
    ignore_nans: False
```

Scalers re-weight the loss

Applied before back-propagation so that every variable, level and grid poin contributes its fair share.

Variable scaling

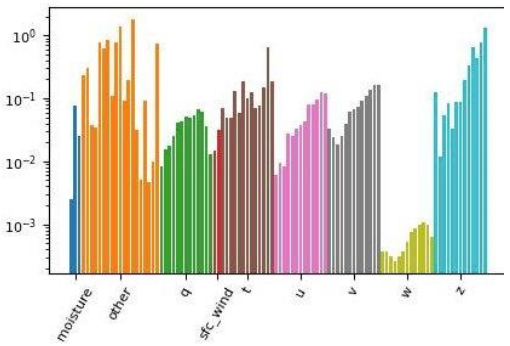
Balance contributions across fields of very different magnitudes.

Pressure-level scaling

Weight the vertical so each level is represented appropriately.

Node-area weighting

Account for unequal grid-cell areas — a pole point ≠ an equator point.



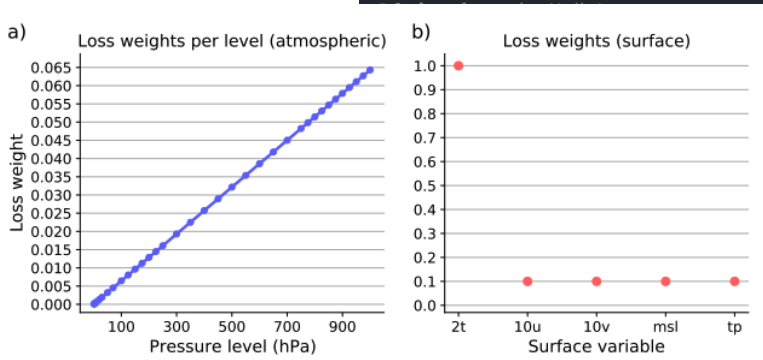

```yaml
general_variable:
  _target_: anemoi.training.losses.scalers.GeneralVariableLossScaler
  weights:
    default: 1
    q: 0.6 #1
    t: 6   #1
    u: 0.8 #0.5
    v: 0.5 #0.33
    w: 0.001
    z: 12  #1
    sp: 10
    10u: 0.1
    10v: 0.1
    2d: 0.5
    tp: 0.025
    cp: 0.0025

pressure_level:
  _target_: anemoi.training.losses.scalers.ReluVariableLevelScaler
  group: pl
  y_intercept: 0.2
  slope: 0.001

# mask NaNs with zeros in the loss function
nan_mask_weights:
  _target_: anemoi.training.losses.scalers.NaNMaskScaler

# tendency scalers
# scale the prognostic losses by the stdev of the variable tendencies (
# useful if including slow vs fast evolving variables in the training (
# if using this option 'variable_loss_scalings' should all be set close
stdev_tendency:
  _target_: anemoi.training.losses.scalers.StdevTendencyScaler

var_tendency:
  _target_: anemoi.training.losses.scalers.VarTendencyScaler
```



$$\mathcal{L}_{\text{MSE}} = \underbrace{\frac{1}{|D_{\text{batch}}|}\sum_{d_0 \in D_{\text{batch}}}}_{\text{forecast date-time}} \underbrace{\frac{1}{T_{\text{train}}}\sum_{\tau \in 1:T_{\text{train}}}}_{\text{lead time}} \underbrace{\frac{1}{|G_{0.25^\circ}|}\sum_{i \in G_{0.25^\circ}}}_{\text{spatial location}} \underbrace{\sum_{j \in J}}_{\text{variable-level}} s_j w_j a_i \underbrace{(\hat{x}_{i,j}^{d_0+\tau} - x_{i,j}^{d_0+\tau})^2}_{\text{squared error}}$$

Schemas

Bringing schema and sanity to your data

Why validate configs?

Runs are controlled by YAML files. A misconfigured field can **silently break a run** — or produce inconsistent results that look fine.

Consistency

All configs follow the same structure and naming.

Error prevention

Catches typos, missing fields and wrong types **before** training starts.

Transparency

Users and automated systems know exactly what parameters are valid.

Validate a config file

```
>>> anemoi-training config validate --config-name my_config

✓ configuration is valid
```

Scaling your models

From one GPU to many — faster training and larger models.

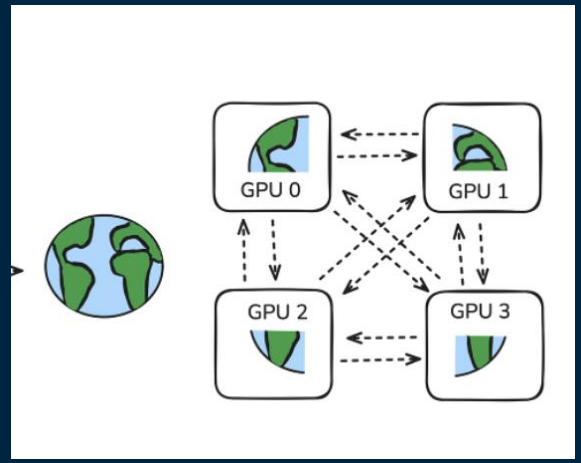
Why do we parallelise?

Faster training

- Share the work across GPUs — e.g. one attention head per GPU
- Each computes a fraction in parallel
- …and synchronise results via messages at the end

Larger models

- Split the globe across multiple GPUs
- Each GPU only communicates its boundary conditions
- Never materialise the full globe in one GPU's memory



SCALING

Two kinds of parallelism

DATA PARALLELISM

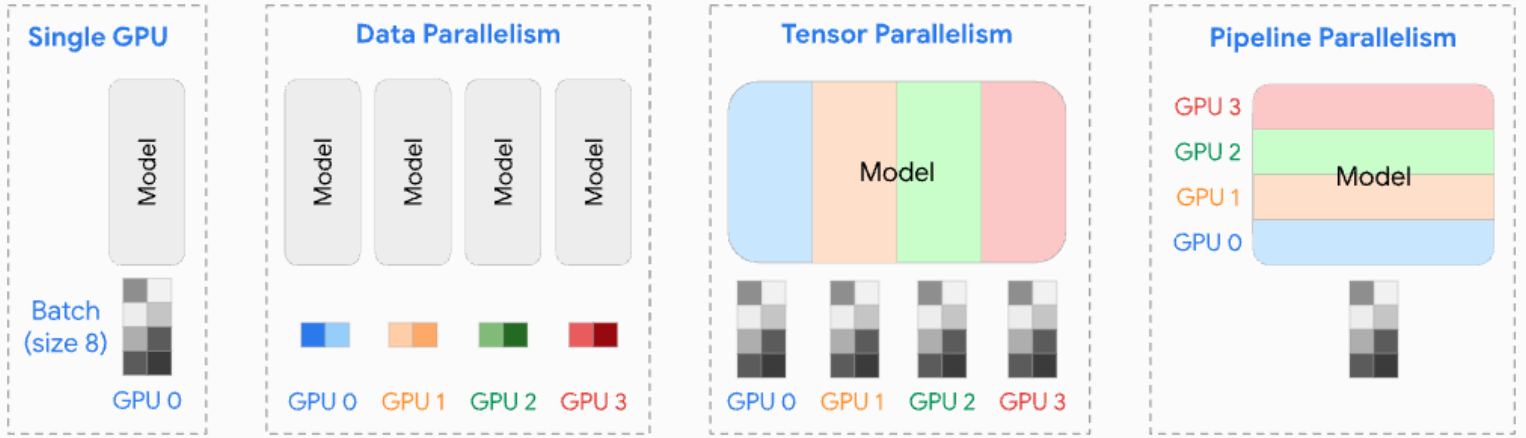
Replicate the model, split the batch

`DistributedDataParallel · PyTorch`

MODEL PARALLELISM

Shard the model across the domain

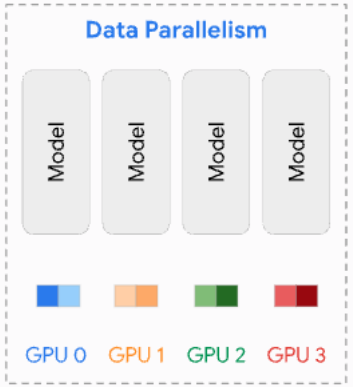
`domain-specific sharding`

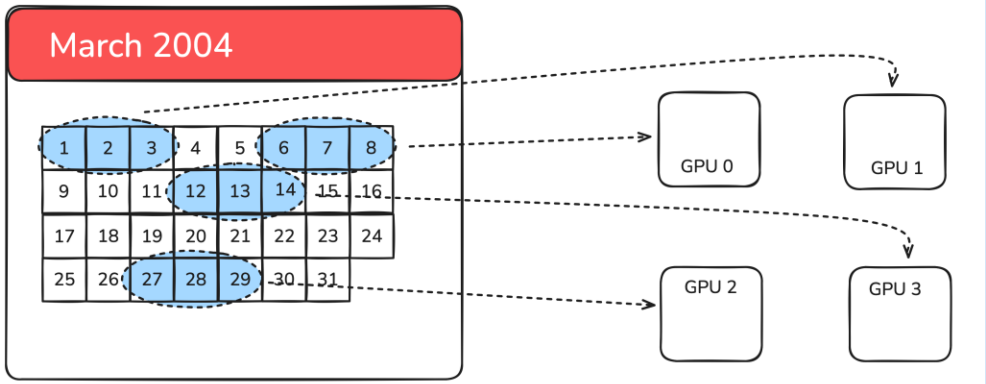


The trade-offs

Data parallelism

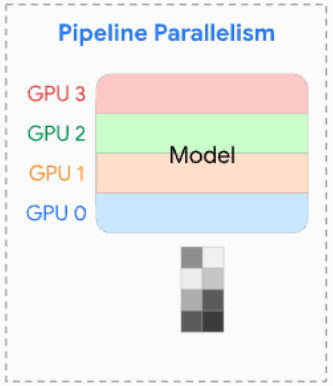
+ Distributes the batch across model replicas; aggregates gradients via all-reduce — **good scaling**
− Limited by batch size

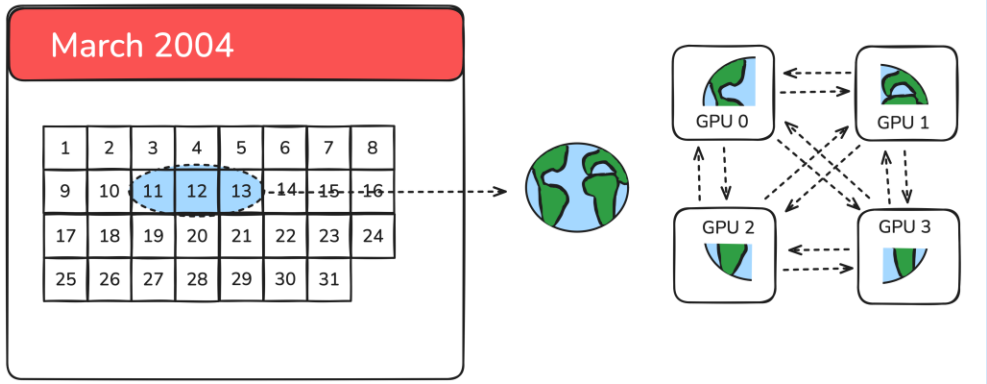




Model parallelism

+ Distributes input data and activations across GPUs; collective communication handles synchronisation
− Limited by communication overhead





Experiment tracking

mlflow

Weights & Biases

Why track experiments?

01 **Reproducibility**
A full record of model versions, parameters, datasets and results

02 **Comparability**
Compare runs — architectures, hyperparameters, datasets — to see what improves skill

03 **Collaboration**
Shared tracking lets teams review and reproduce across institutes

04 **Performance monitoring**
Centralised logging of metrics (RMSE, ACC) over time

05 **Automation & integration**
MLflow fits naturally into Anemoi-Training pipelines — experiment metadata flows into operational workflows

EXPERIMENT TRACKING

A pluggable logger

The Trainer holds a `logger` — swap the backend under `diagnostics/log/`.

`trainer` → `logger` →

mlflow

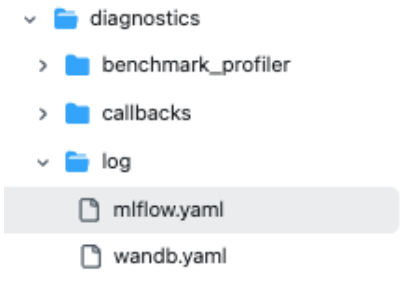
`AnemoiMLflowLogger`

the default backend

Weights & Biases

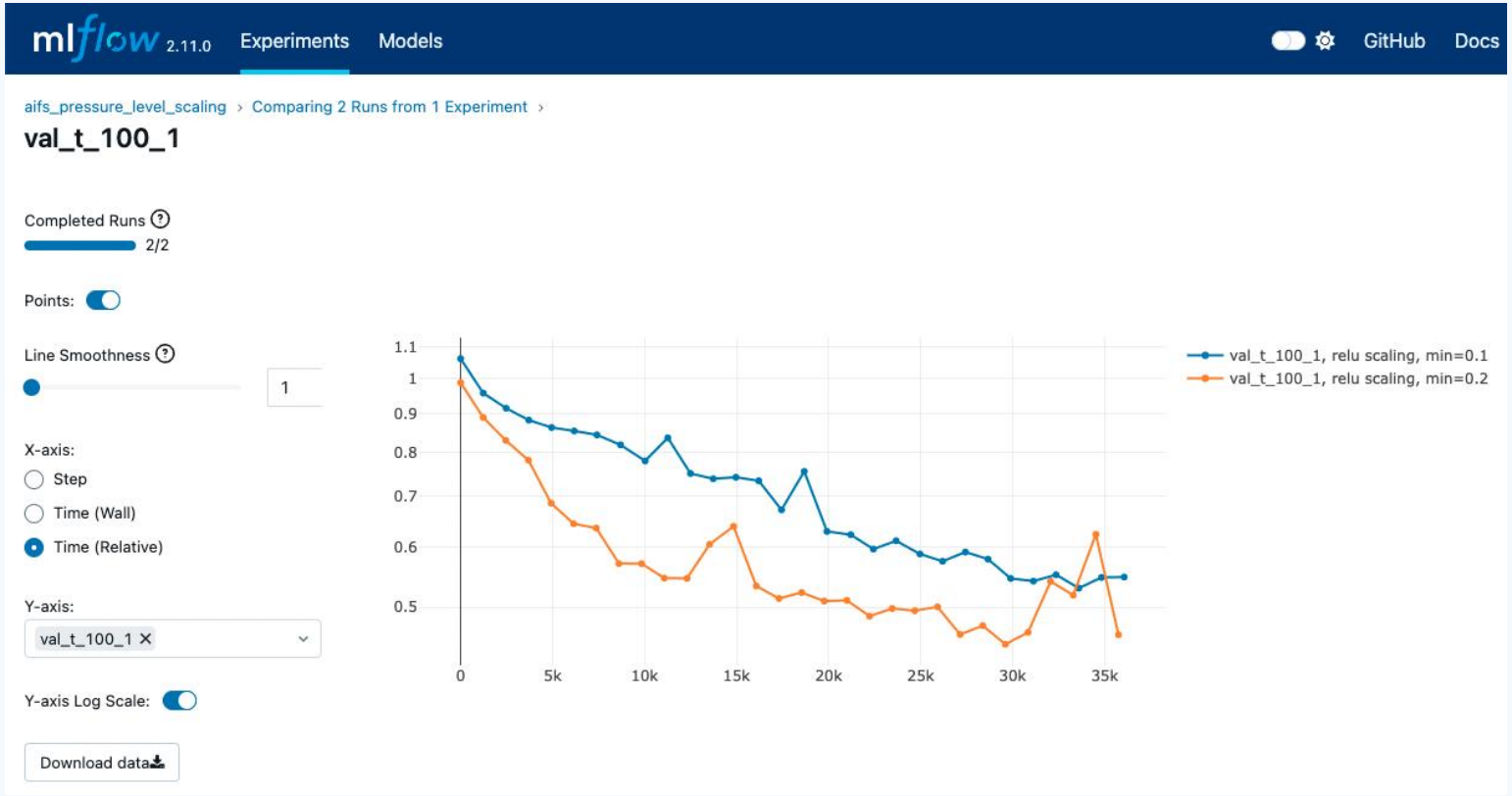
`WandbLogger`

alternative backend



Comparing runs in MLflow

Overlay validation curves across runs to see what improves skill.

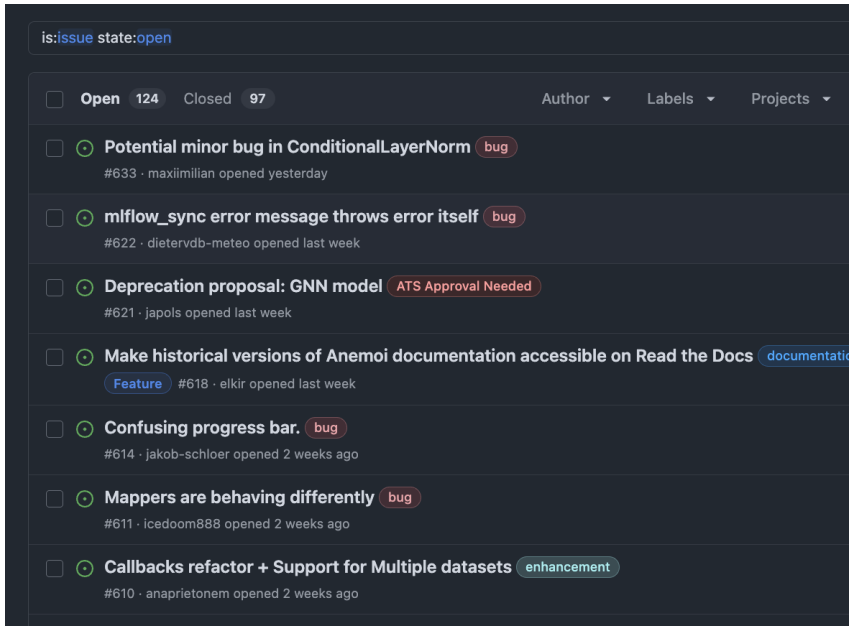


Contributing to anemoi

Anemoi is a community project — issues and pull requests welcome.

Open an issue

1. Go to the repository (e.g. `github.com/ecmwf/anemoi-core`)
2. Issues → "New issue"
3. Choose the right template — bug report or feature request
4. Submit — maintainers triage and respond



FILL IN CLEAR DETAILS

- What happened?
- Steps to reproduce the bug
- Logs and error messages
- Environment — Python version, OS, dependencies

Create a pull request

When: You've fixed a bug, improved documentation, or added a feature.

01 Fork the repository to your account

02 Clone your fork and create a branch

03 Make and test your changes

04 Commit with a clear message

05 Push and open a PR — explain what & why

06 Wait for automated checks & review

A FEATURE NEEDS

- Tests
- Schemas
- Docs

CONTRIBUTING

anemoi-configs

A shared repository of complete, **versioned, reproducible** model configurations.

Reproduce a released model

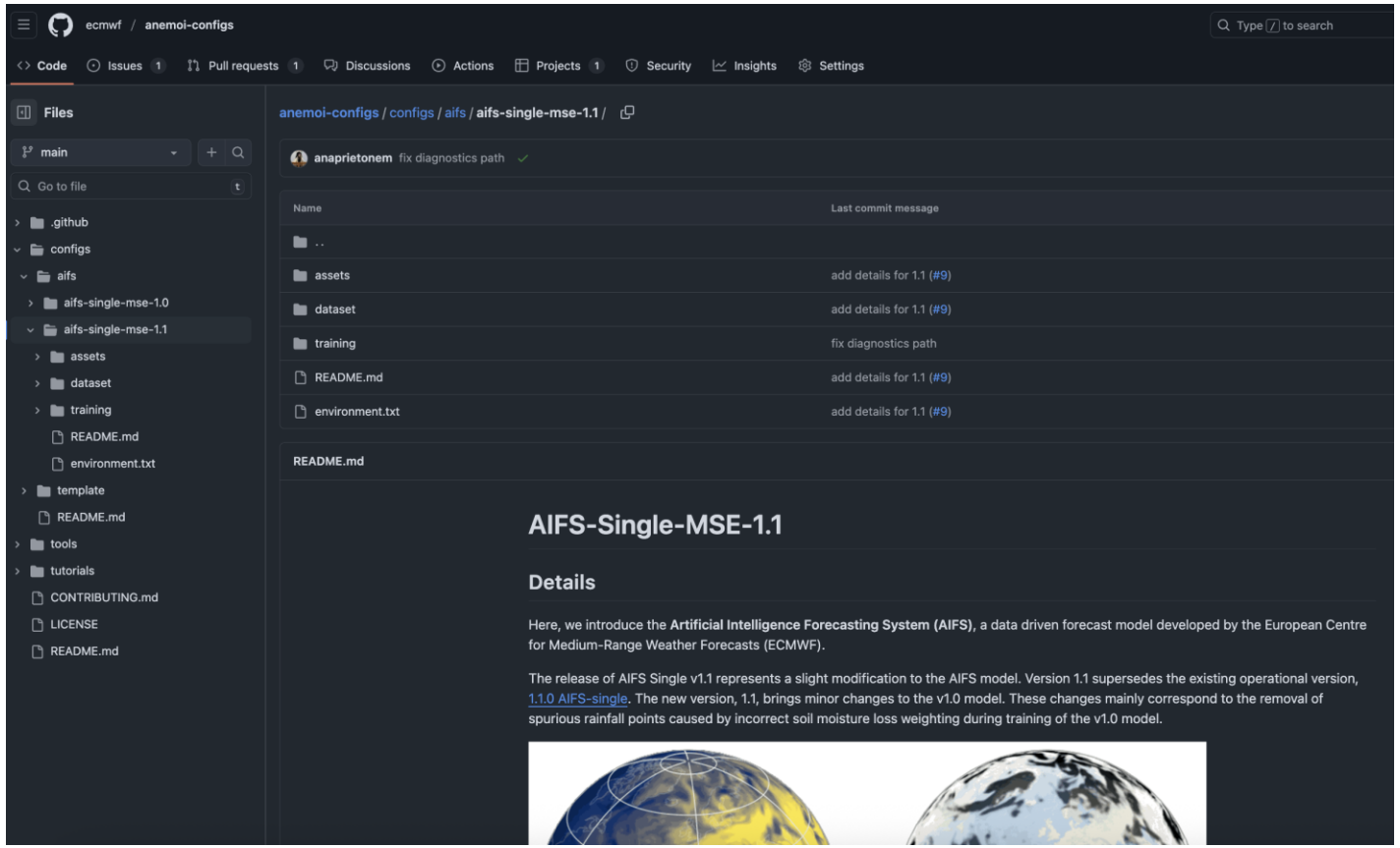
e.g. `AIFS-Single-MSE-1.1` — dataset, training & assets, together.

Start from known-good

Fork a config, adapt it to your data, and run.

Contribute yours back

Share configurations that work — for the whole community.



Useful links

DOCS	anemoi.readthedocs.io/projects/training
CONFIG TEMPLATES	github.com/ecmwf/anemoi-core · training/src/.../config
TEST CONFIGS	github.com/ecmwf/anemoi-core · training/tests/integration

Anemoi Inference

Harrison Cook

The last link in the chain

Training emits a checkpoint. Inference runs it forward — driven by its **own** config, fully decoupled from how the model was trained.

`anemoi-training train`

driven by the **training config**

→ EMITS

ATOMIC ARTEFACT

`checkpoint.ckpt`

→ RUNS

`anemoi-inference run`

driven by a separate **inference config**

DESIGN

Built for operations

A community project where **"any" model** can be run — but with operationalisation always in mind.

Focus on inference
The inference problem, and nothing more.

Minimal dependencies
No unnecessary baggage to install.

Different data sources
Meet data where it already lives.

Limited compute
Runs where resources are tight.

External tooling
Integrates into wider workflows.

Decoupled from training
Independent of how the model was trained.

CAPABILITIES

Features

Automatic discovery

Model parameters read from metadata embedded in the checkpoint — prognostic, diagnostic and forcing variables. Minimal-to-no config to retrieve initial conditions.

User-extendable I/O

Multi-format input and output — e.g. `grib` in, `netcdf` out. Add your own.

Decoupled from training

Inference input is completely independent of how — and on what — the model was trained.

Parallel running

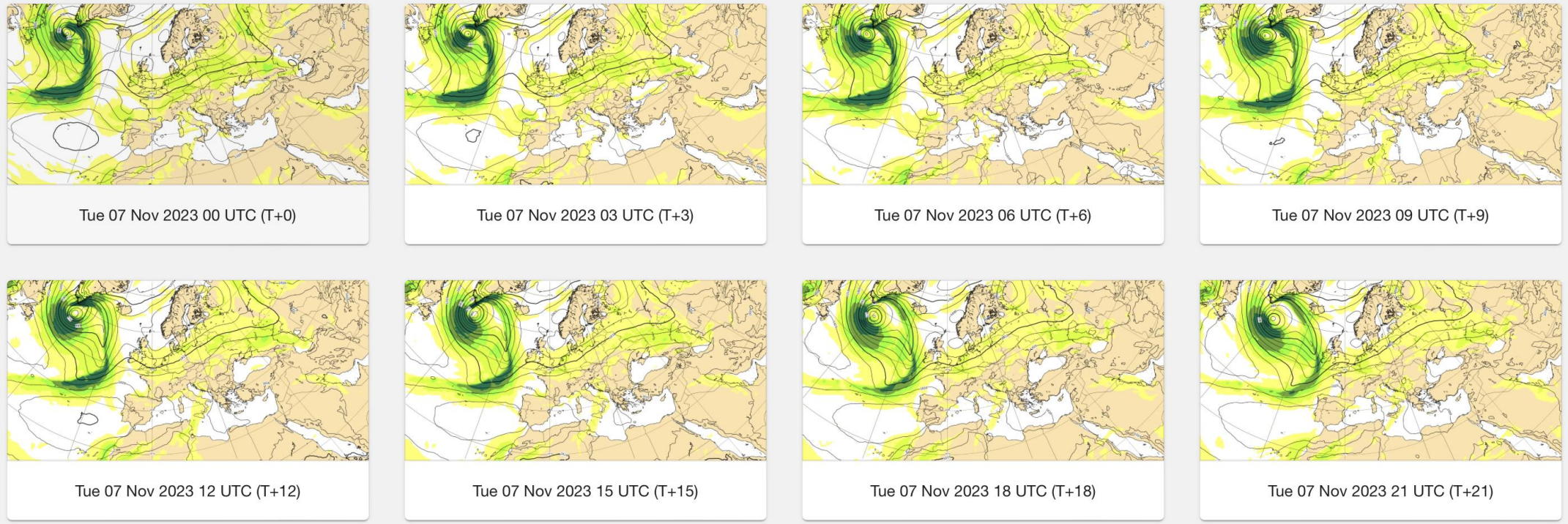
For large models — across Slurm, Azure and other clusters.

Autoregressive rollout

A model that predicts **t+6** reaches any multiple of 6 by feeding its own output back in.

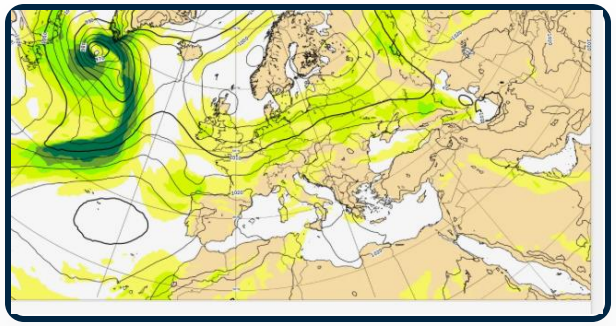
`t` —6h→ `t+6` —6h→ `t+12` —6h→ `t+18` ··· `t+240`

predict → feed back → predict



Temporal interpolation

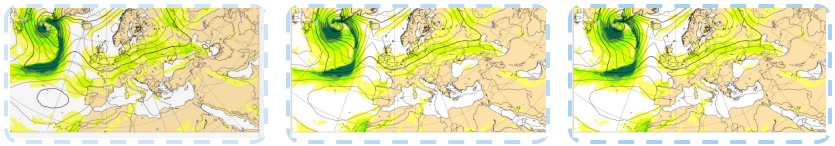
Given two states 6 hours apart, predict the steps in between — densifying a forecast in time.



`t · known`

→

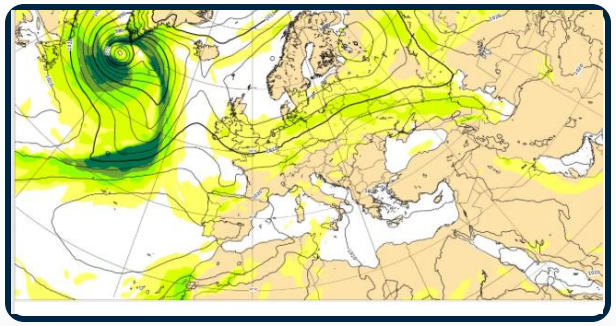
PREDICTED IN BETWEEN



t+1 t+3 t+5

interpolated frames

→



`t+6 · known`

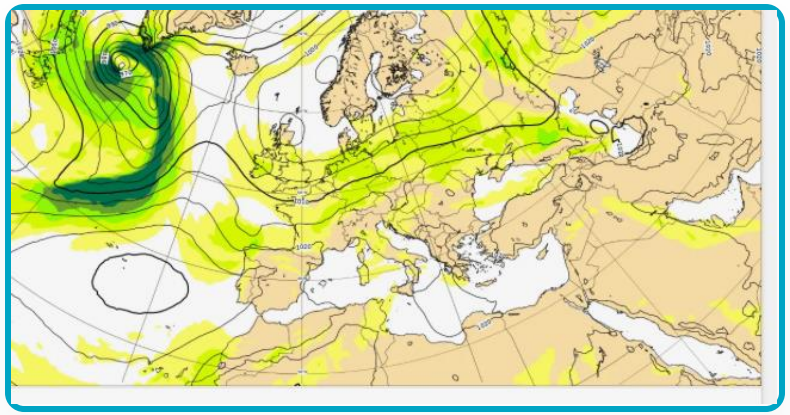
Downscaling

COMING SOON

Same time, higher resolution — map a coarse state to a fine one, no time-stepping involved.

coarse input
low resolution

→ DOWNSCALE

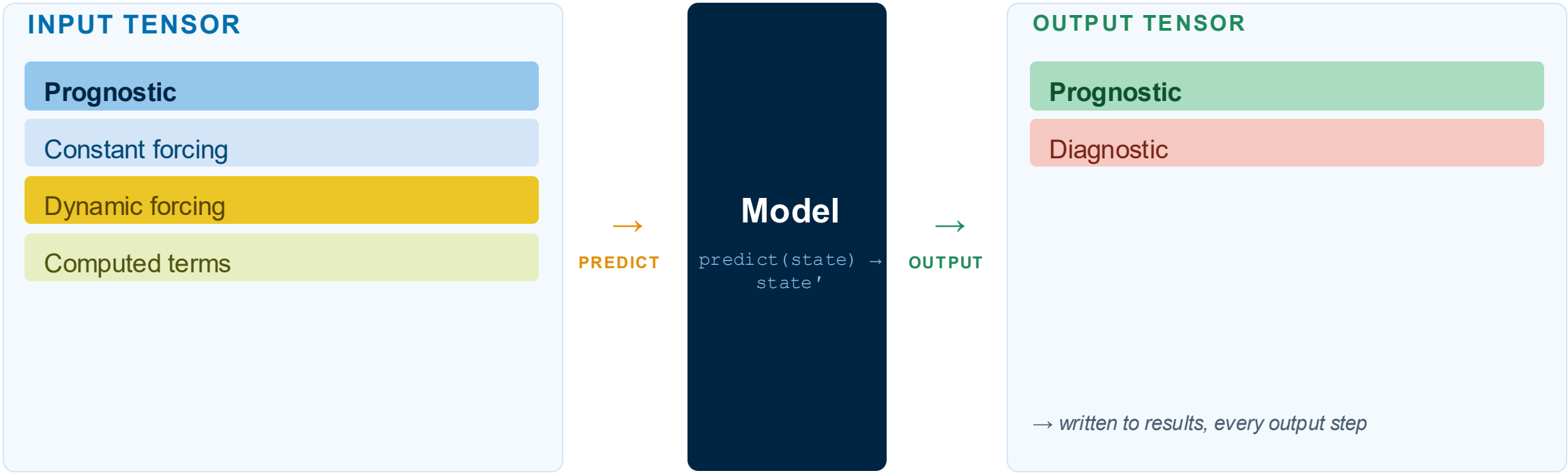


fine output
high resolution · same time

THE MENTAL MODEL

Input → Model → Output

State flows through three stages, with forcings injected along the way. Prognostic outputs are copied back to seed the next step.



copy Prognostic outputs are copied back into the next input tensor; forcings are re-supplied — then the model steps forward again.

THE MENTAL MODEL

Every variable has a category

prognostic	Input *and* output — the model evolves these
diagnostic	Output only, derived by the model (total precip, cloud cover)
forcing	Imposed from outside the model during the run
constant	Static fields that never change (orography, land-sea mask)
computed	Calculated at run time (cos_julian_day, solar terms)
accumulation	Accumulated over time (e.g. precipitation)

NOTE

A variable can hold **several categories at once**.

LOOK INSIDE THE BOX

```
$ anemoi-inference inspect --variables checkpoint.ckpt

prognostic: 2t, 10u, 10v, z…
diagnostic: tp, tcc
forcing : lsm, z (orog)
```

INTERFACES

Three API levels

Pick the level of control your use case needs.

LOW LEVEL

NumPy → NumPy

Arrays in, arrays out. Maximum control, for embedding in custom code.

HIGH LEVEL

Object-oriented

Runner, inputs, outputs, processors — for building applications on inference

EVERYDAY

Command line

One YAML config, one command. Our focus from here.

COMMAND LINE

Run it from one command

```
$ anemoi-inference run config.yaml
```

INFERENCE

run	Run a forecast from a config
couple	Coupled tasks (e.g. atmosphere–ocean)
retrieve	Generate the data-retrieval requests a config needs

CHECKPOINT MANAGEMENT

inspect	look inside	**metadata**	extract arrays
validate	check integrity	**patch**	edit metadata
sanitise	remove paths	**requests**	show MARS reqs

CONFIGURATION

One YAML drives everything

```yaml
checkpoint: /path/to/model.ckpt
lead_time: 240 # hours
date: 2026-05-30T00

input:
  mars: {}

post_processors:
  - accumulate_from_start_of_forecast

output:
  tee:
  - grib: output.grib
  - plot: {domain: Europe}
```

THE PATTERN

The first key names the **Python class** to use; the rest are its arguments.

It repeats across inputs, outputs and processors.

IT COLLAPSES GRACEFULLY

```yaml
input: mars # defaults

input: {grib: file.grib} # one arg

input: # full nested block
```

Metadata travels with the model

A trained model is stored in an **atomic file** — weights plus everything inference needs to set itself up.

METADATA INSIDE

- Datasets used for training
- Operations applied to the data (e.g. `cutout`)
- Grid & area
- Mapping of variables to positions in the input tensor

SO INFERENCE CAN…

- Set up its pipeline **automatically**
- Know the prognostic, diagnostic & forcing variables
- Need minimal-to-no config to retrieve initial conditions

INPUTS

Meet data where it lives

Initial conditions from many sources — pick the one your organisation can access.

mars
ECMWF archive

cds
Copernicus CDS / ERA5

opendata*
open IFS analysis

polytope*
regional extractions

datasets
your anemoi-datasets

grib
any local GRIB file

netcdf
CF-convention files

dummy
fake input to test

cutout
nested LAM — limited-area + global

* available as a plugin

PROCESSORS

Shape data in and out

Data is weird — it comes in many shapes and sizes. Processors transform state around the model.

Pre-processors

Prepare the initial conditions before the model sees them.

anemoi-transforms

M

MODEL

Post-processors

Finalise the output state.

`accumulate_from_start_of_forecast` `extract_from_state` `extract_mask` `assign_mask`

OUTPUTS

Write what you need

Writing is often the operational bottleneck — outputs are pluggable and support parallel I/O.

printer
min/max per field · default

grib
templated filenames

netcdf
array format

zarr
Zarr 2 & 3

plot
maps via earthkit-plots

raw
npz, for debugging

tee
several at once

SHARED PARAMS `variables output_frequency post_processors write_initial_state`

PARALLEL OUTPUT Different runners each write a **subset of the data** — removing the write bottleneck in operations.

Scale beyond one device

Set `runner: parallel` and the engine distributes the model across GPUs — auto-detecting the cluster.

AUTO-DETECTED LAUNCHERS

Slurm	SLURM_NTASKS + SLURM_JOB_NAME	**srun**
MPI	OMPI_COMM_WORLD_SIZE / PMI_SIZE	**mpirun**
torchrun	RANK + LOCAL_RANK	**torchrun**
manual	specify world_size yourself	—

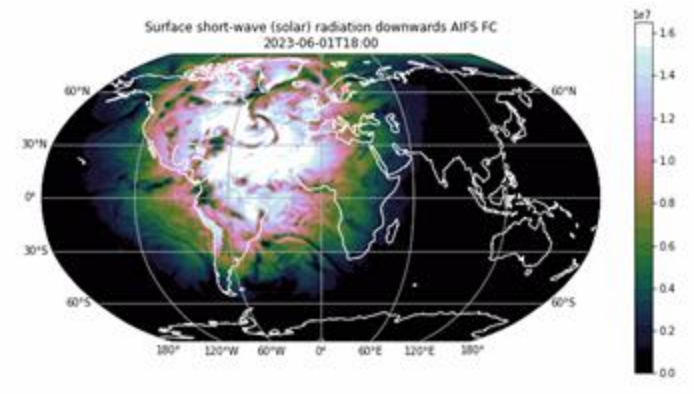
GOOD TO KNOW

- By default **rank 0** writes; with **parallel output** each runner writes its own subset
- Tune memory via chunking — `ANEMOI_INFERENCE_NUM_CHUNKS`
- Requires **anemoi-models ≥ 0.4.2**

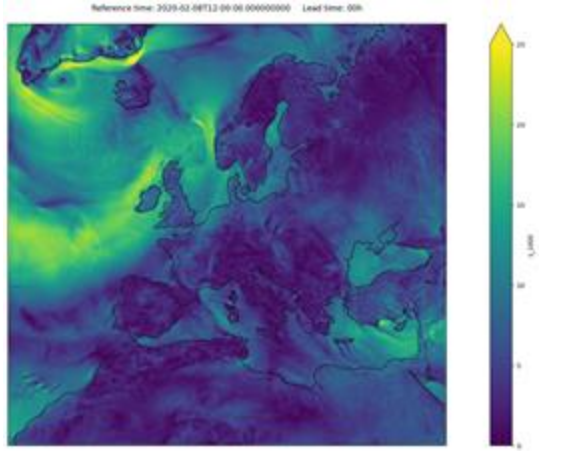
What does Anemoi enable?

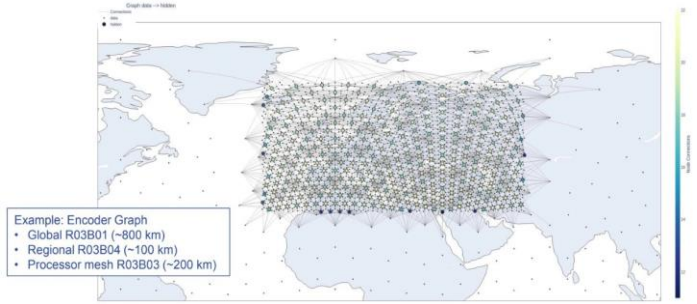
Inference

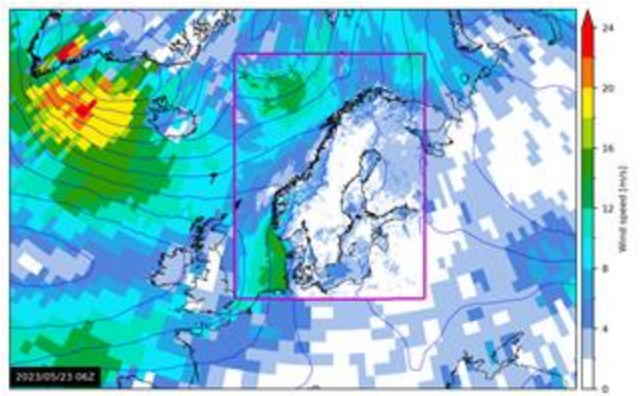
Global

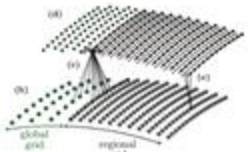


Regional Area Modelling









Nipen et al. 2024

Stretched Grid Model

DISCOVER ANEMOI · WEBINAR SERIES · PART 3

Thank you

Questions?

www.ecmwf.int

PRESENTER
Ana Prieto Nemesio
Machine Learning Team Lead
ana.prietonemesio@ecmwf.int

PRESENTER
Harrison Cook
Research Software Engineer
harrison.cook@ecmwf.int