Anemoi Training

Machine Learning and Destination Earth training course

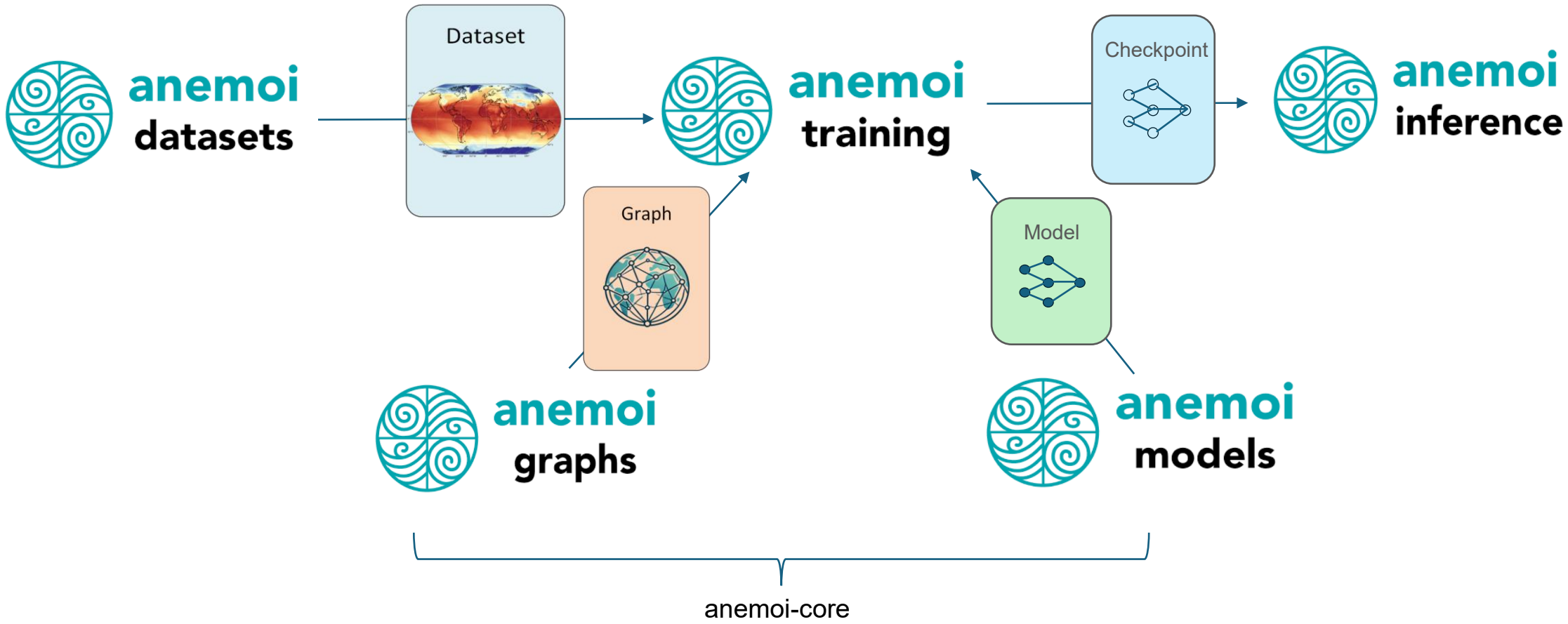
Ana Prieto Nemesio

ana.prietonemesio@ecmwf.int

Overview

- What's anemoi-training and what you can do with it
- How to get started with anemoi-training
- Running anemoi-training = configuration.
- Keep learning about anemoi-training

Anemoi: An overview of its main components



What's anemoi-training about?

- Anemoi-training is a python package designed to allow users and developers to train machine learning models for weather forecasting.
- Some of the key features supported by anemoi-training are:

  - Code to train models, using pytorch-lightning and Hydra
  - Support for autoregressive and multi-input & output forecasting.
  - Multi-node/multi-GPU support + efficient data and model parallelisation strategies
  - Deterministic / probabilistic training + Downscaling + Autoencoder
  - Callbacks for profiling, evaluating, plotting and logging intermediate results
  - Experiment tracking with e.g. mlflow
  - Customised loss functions suitable for weather forecasting and variable scaling

What’s anemoi-training about?

- What can't you do with it?
  - It’s not a general-purpose deep learning framework – it’s specialised in graph neural networks.
  - You can’t train vision models with anemoi-training
  - Is designed for multidimensional spatio-temporal modelling, not single-point temporal forecasting.

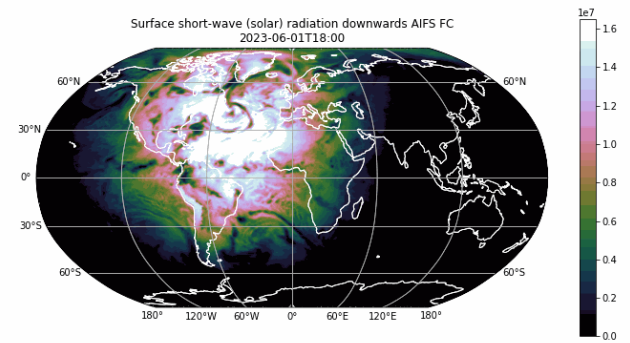
- Who's this useful for?
  - 🌍 Weather & climate ML scientist and engineers
  - 🧠 Researches that want to build data-driven weather forecasting models
  - 🏢 Institutions looking to develop its own operational DDWF models

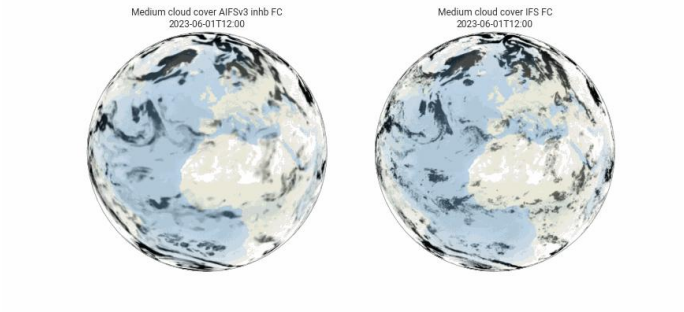
Example of what you can do with anemoi-training

Forecasting

Deterministic Models

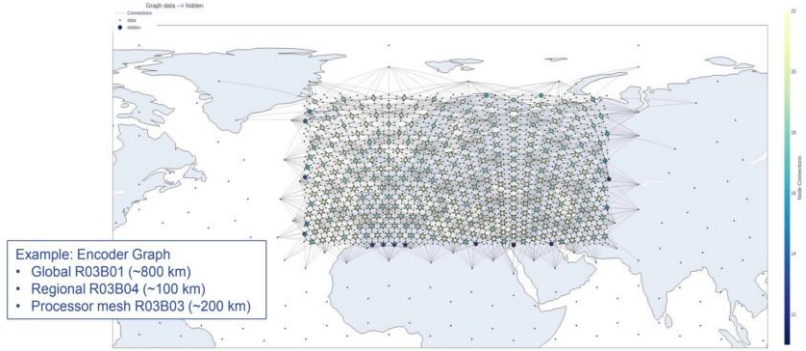
Global Model



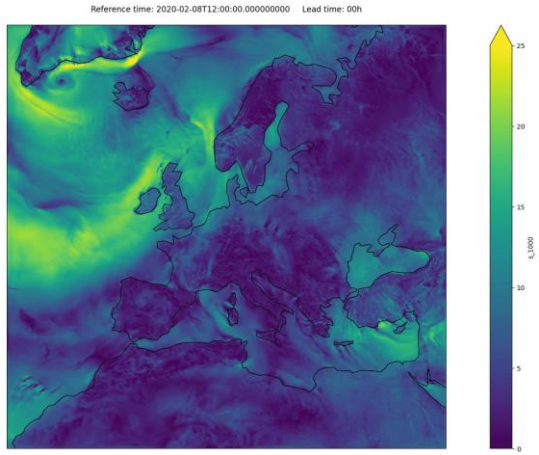


AIFS v1

LAM

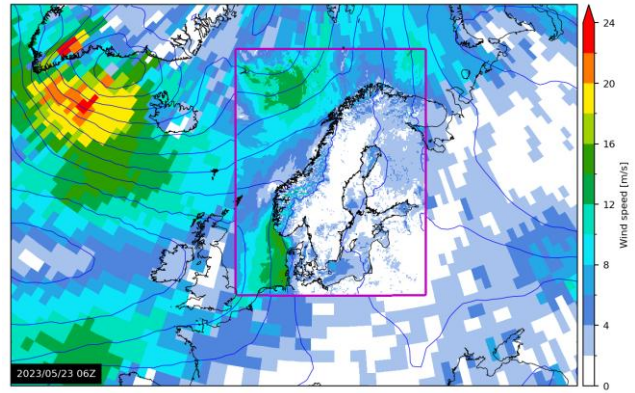


AICON LAM - Encoder graph

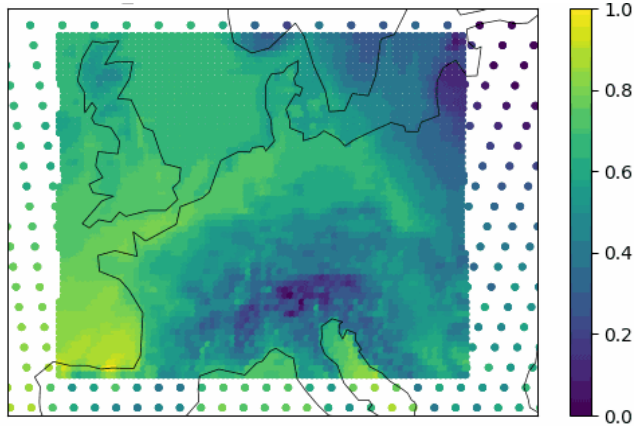


RMI CERRA LAM at 5.5km

Stretched Grid



MetNorway - Stretched Grid Bris + Temporal Interpolation



KNMI Stretched Grid CERRA
*coarser resolution

Examples of models you can train with anemoi-training

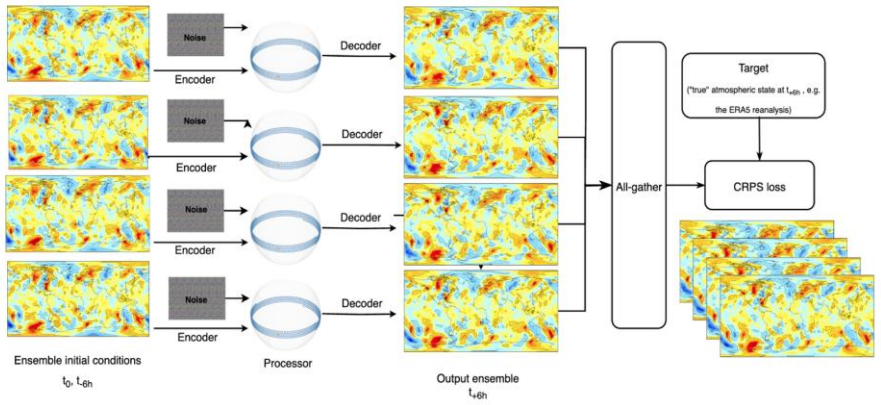
Forecasting

Probabilistic Models

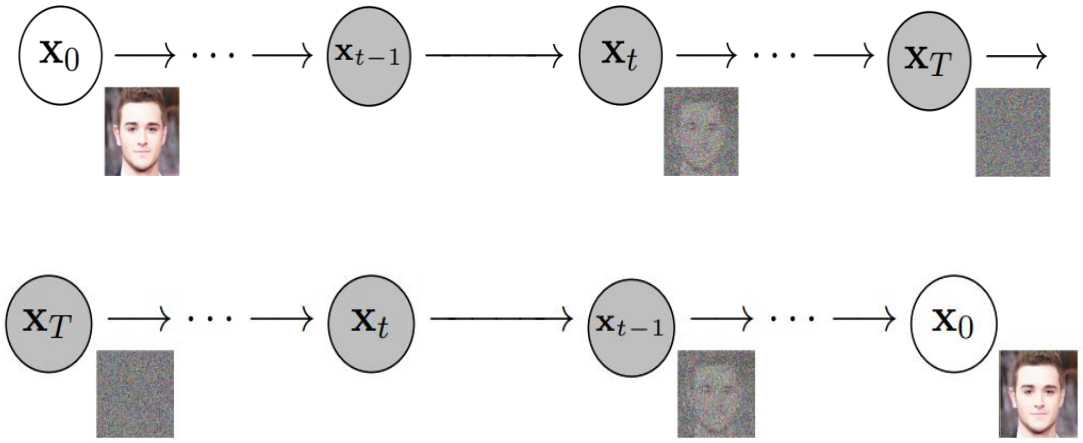
LAM

Stretched Grid

Global Model



Lang et al. 2024b

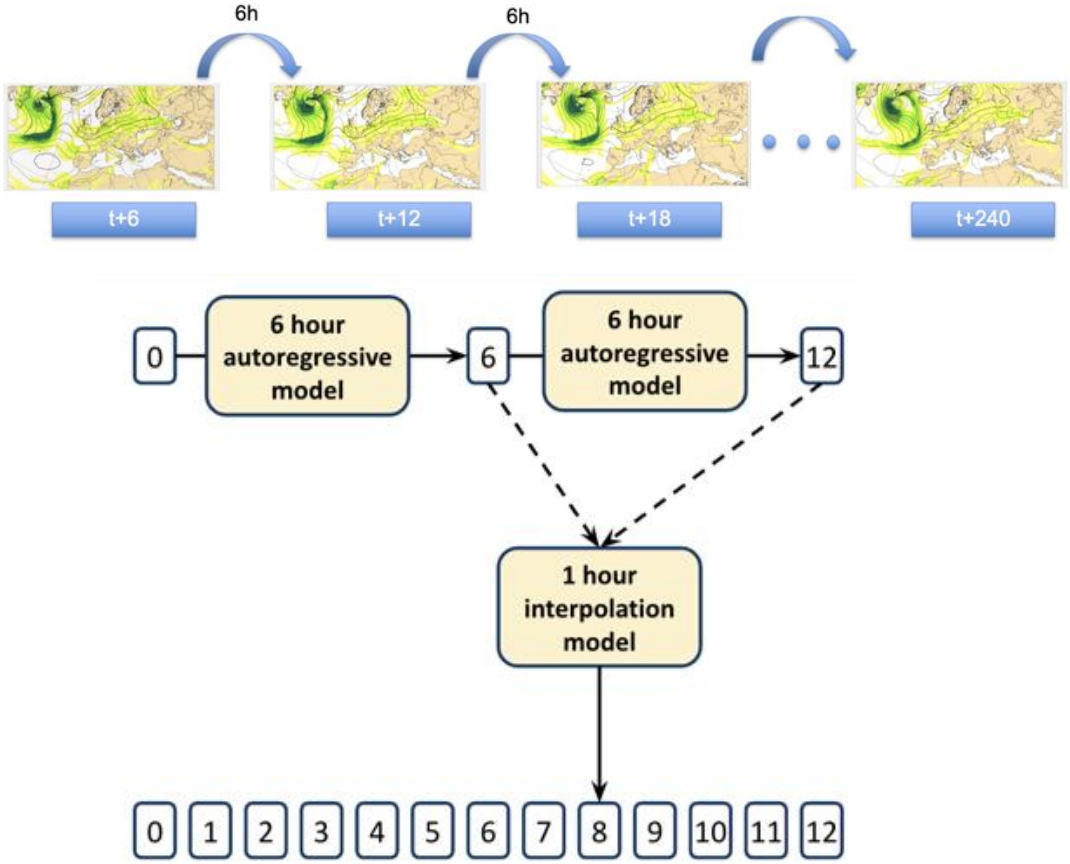


CRPS-based

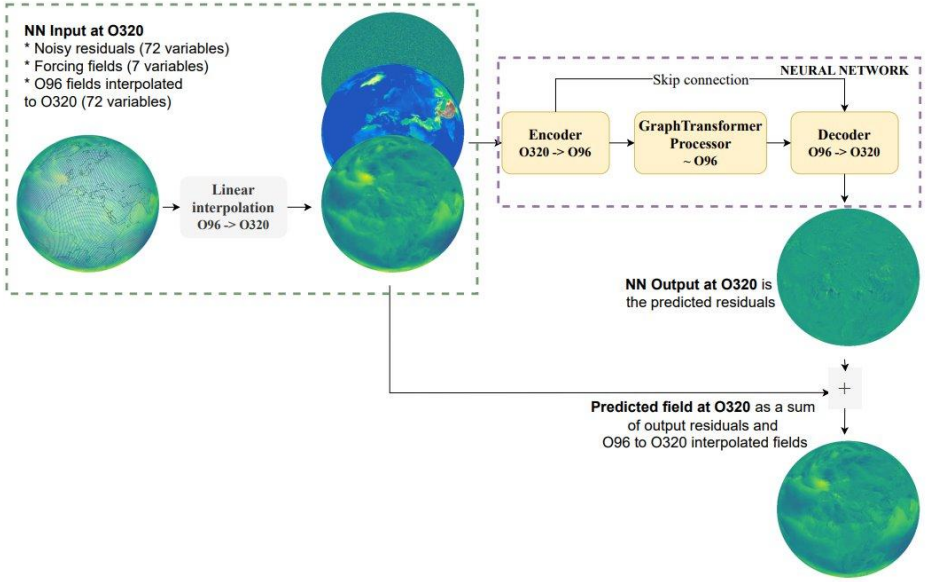
Diffusion Model

Examples of models you can train with anemoi-training

Temporal Downscaling



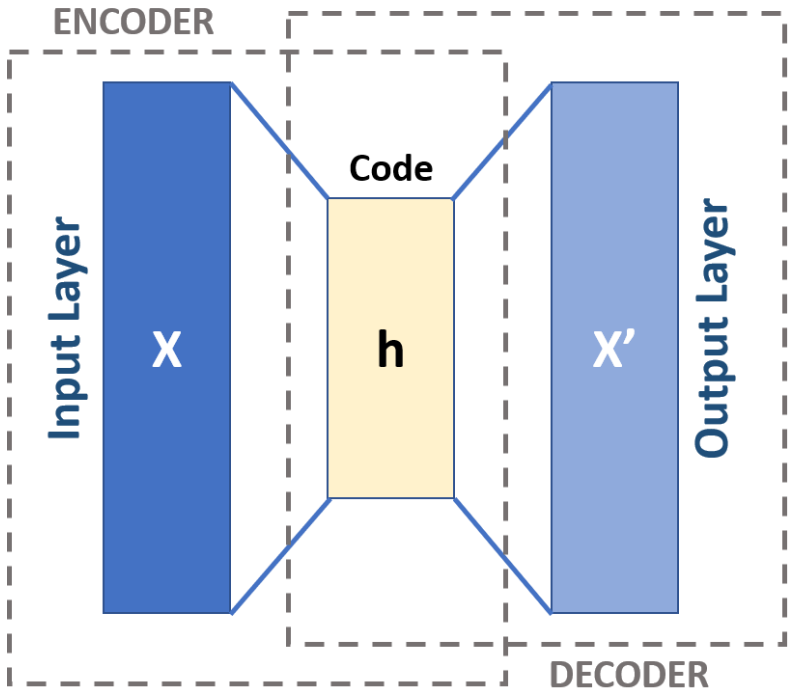
Spatial Downscaling



ECMWF

Examples of models you can train with anemoi-training

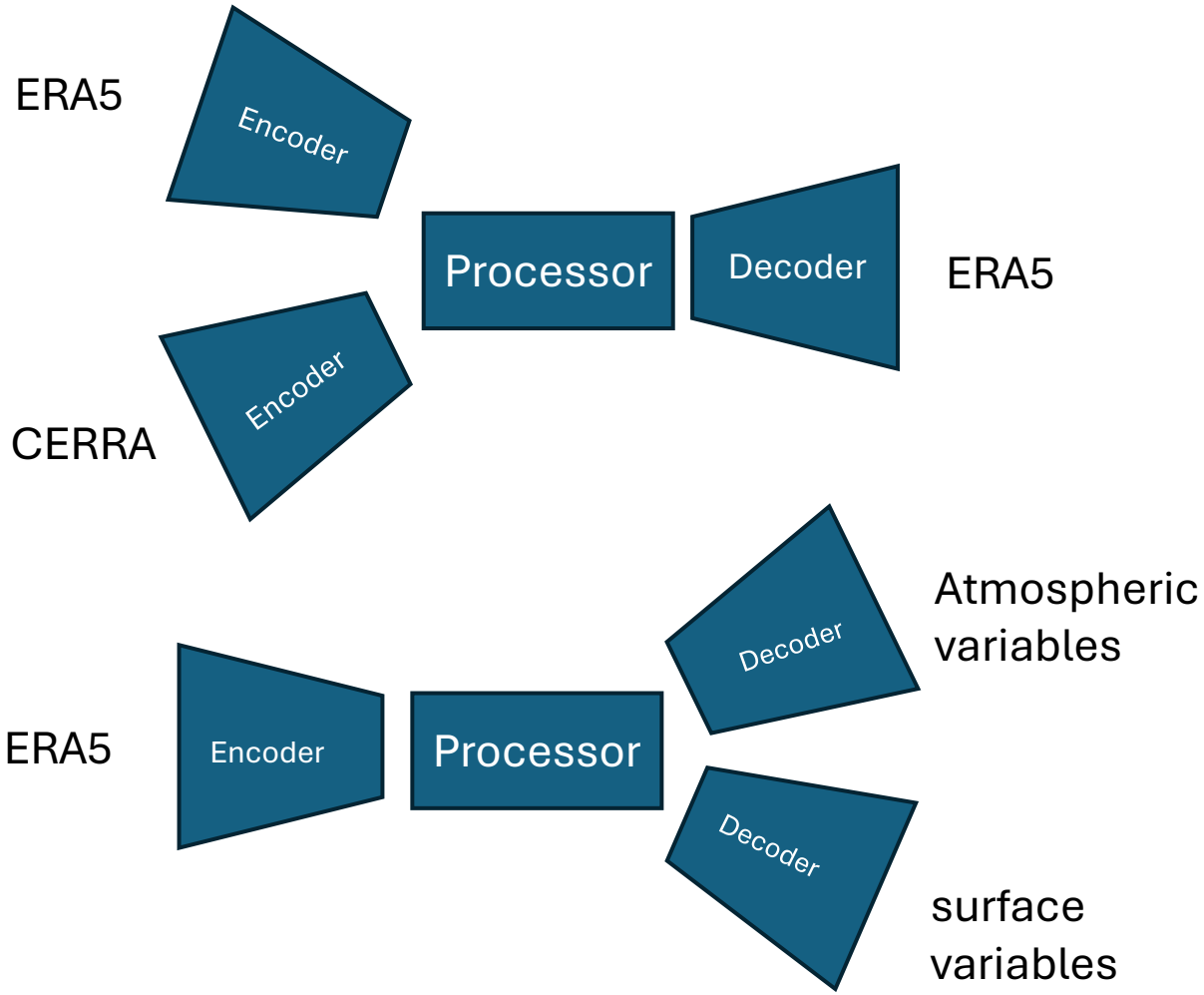
Autoencoder



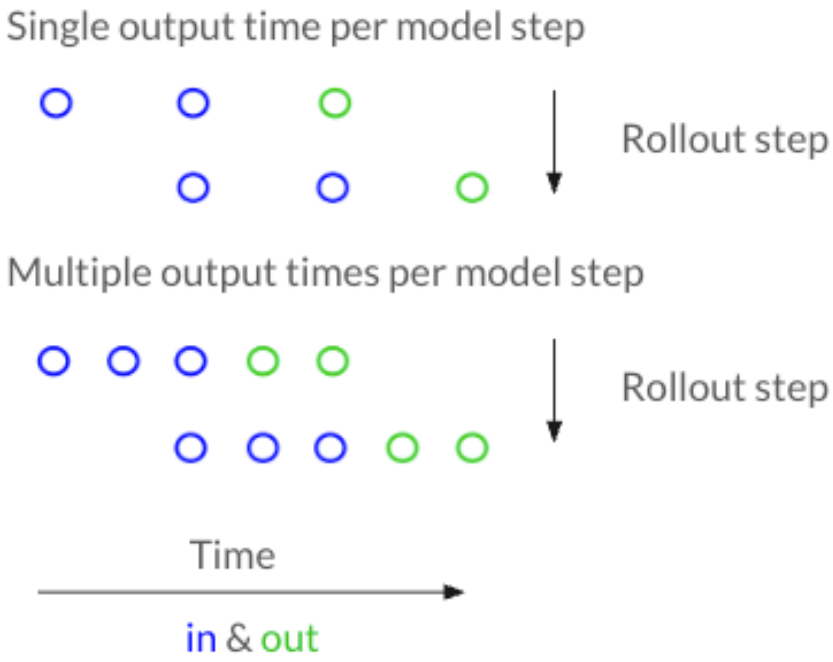
- Enable scalable latent-space modelling
- Support superresolution and reconstruction tasks
- Learn compressed spatial representations

Examples of models you can train with anemoi-training

Multiple-datasets

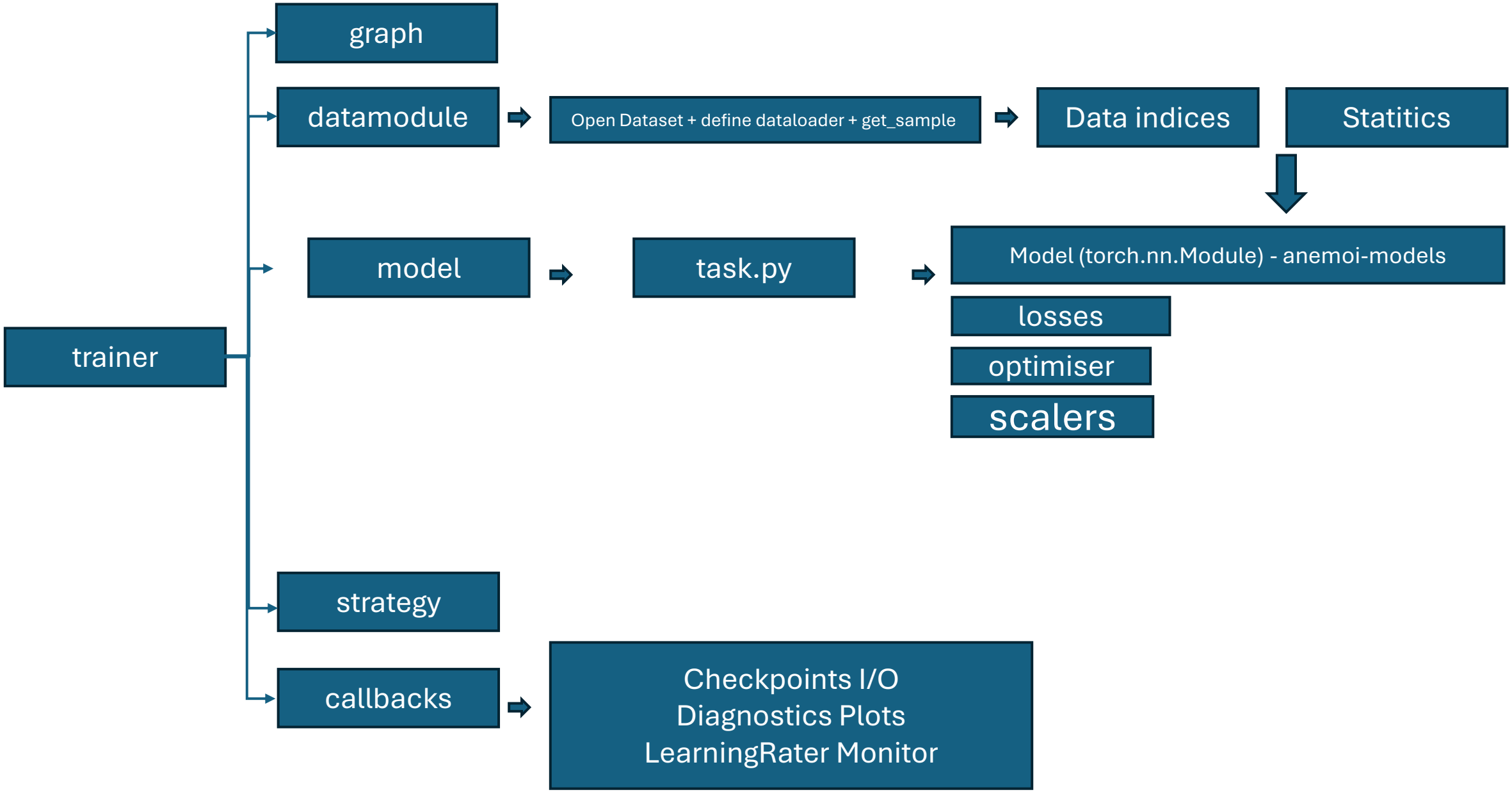


Multiple-output

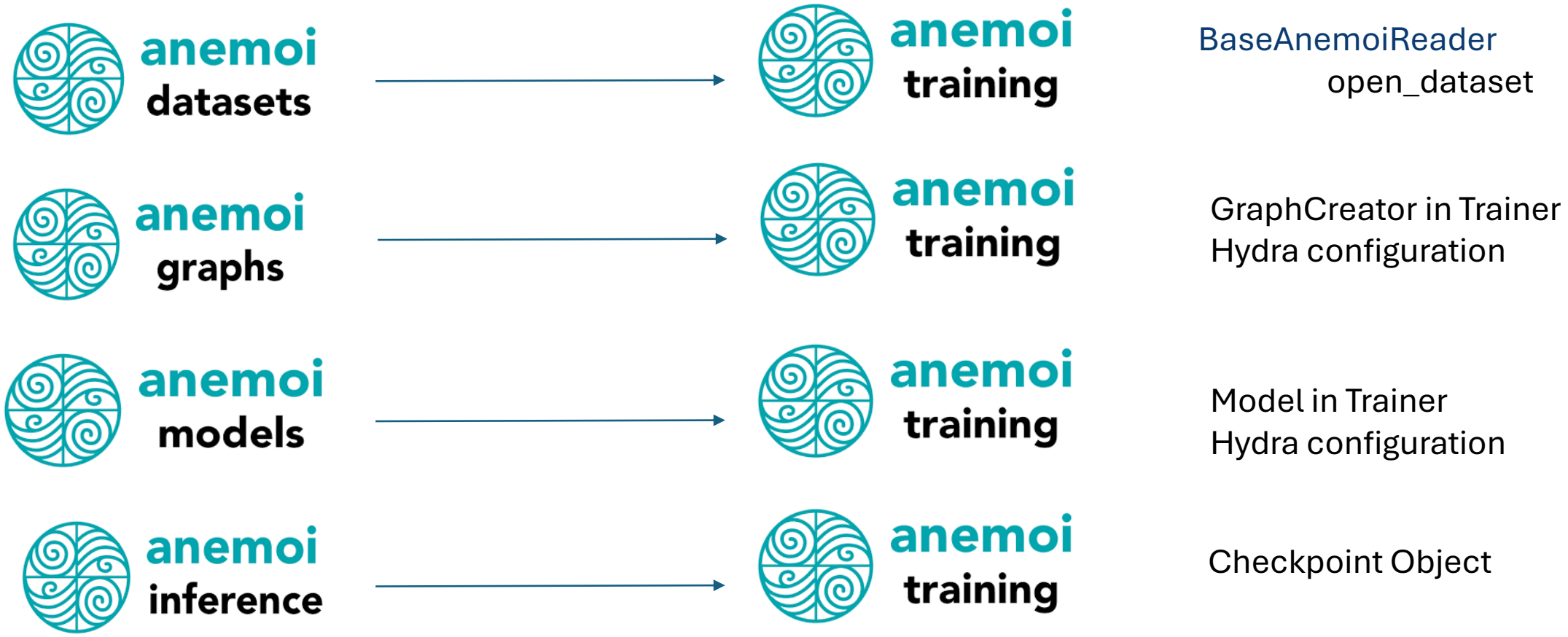


ECMWF

Getting started with anemoi-training


graph
datamodule
Open Dataset + define dataloader + get_sample
Data indices
Statitics
model
task.py
Model (torch.nn.Module) - anemoi-models
losses
optimiser
scalers
trainer
strategy
callbacks
Checkpoints I/O
Diagnostics Plots
LearningRater Monitor

Anemoi: Interfaces Overview



Anemoi: Interfaces Overview

```python
class BaseAnemoiReader:
    """Anemoi data reader for native grid datasets."""

    def __init__(
        self,
        dataset: str | dict | None = None,
        dataset_config: str | dict | None = None,
        start: datetime.datetime | int | None = None,
        end: datetime.datetime | int | None = None,
    ):
        """Initialize Anemoi data reader."""
        source = dataset_config if dataset_config is not
        if source is None:
            msg = "Either dataset or dataset_config must
            raise ValueError(msg)
        self.data = open_dataset(_normalize_dataset_confi
```

```python
@cached_property
def model(self) -> pl.LightningModule:
    """Provide the model instance."""
    assert (
        not (
            "GLU" in self.config.model.processor.layer_kernels["Activation"]["_target
            and ".Transformer" in self.config.model.processor.target_
        )
        and not (
            "GLU" in self.config.model.encoder.layer_kernels["Activation"]["_target_"
            and ".Transformer" in self.config.model.encoder.target_
        )
        and not (
            "GLU" in self.config.model.decoder.layer_kernels["Activation"]["_target_"
            and ".Transformer" in self.config.model.decoder.target_
        )
    ), "GLU activation function is not supported in Transformer models, due to fixed
    "Please use a different activation function."

    kwargs = {
        "config": self.config,
        "data_indices": self.data_indices,
        "graph_data": self.graph_data,
        "metadata": self.metadata,
        "statistics": self.datamodule.statistics,
        "statistics_tendencies": self.datamodule.statistics_tendencies,
        "supporting_arrays": self.supporting_arrays,
    }

    model_task = get_class(self.config.training.model_task)
    model = model_task(**kwargs)  # GraphForecaster -> pl.LightningModule
```

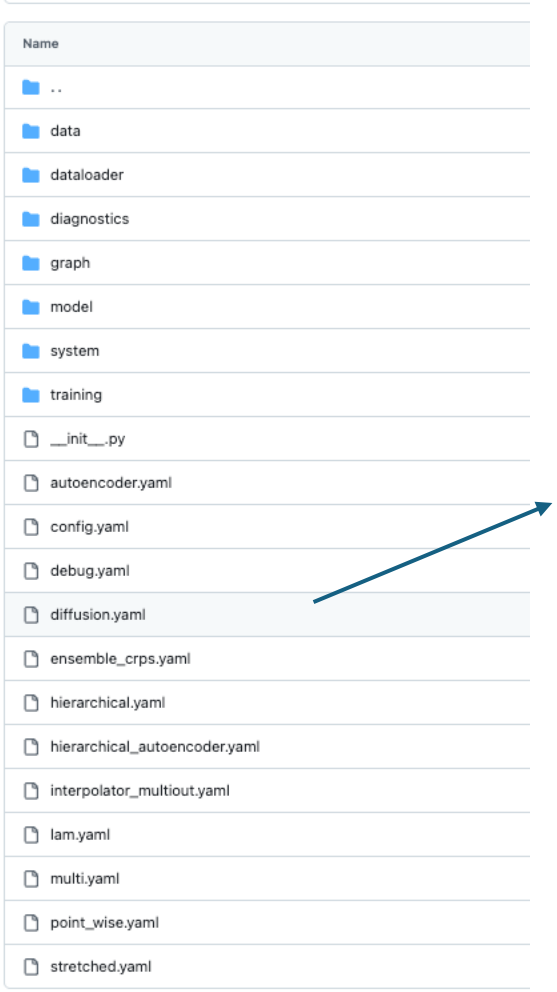
anemoi-core / training / src / anemoi / training / train / **train.py**

Code | Blame | 596 lines (499 loc) · 25.1 KB

```python
146     def _create_graph_for_dataset(self, dataset_path: str, datas
146     def _create_graph_for_dataset(self, dataset_path: str, datas
147         """Create graph for a specific dataset, overriding the c
148         # Determine filename
149         if (graph_filename := self.config.system.input.graph) is
150             graph_filename = Path(graph_filename)
151             if graph_filename.name.endswith(".pt"):
152                 graph_name = graph_filename.name.replace(".pt",
153                 graph_filename = graph_filename.parent / graph_r
154
155             # Try loading existing
156             if graph_filename.exists() and not self.config.graph
157                 from anemoi.graphs.utils import get_distributed_
158
159                 LOGGER.info("Loading graph data from %s", graph_
160                 return torch.load(graph_filename, map_location=g
161         else:
162             graph_filename = None
163
164         # Create new graph
165         from anemoi.graphs.create import GraphCreator
166
167         graph_config = self.config.graph
168
169         # ALWAYS override dataset from dataloader config (ignore
170         if hasattr(graph_config.nodes, "data") and hasattr(graph
171             graph_config.nodes.data.node_builder.dataset = datas
172
173         return GraphCreator(config=graph_config).create(
174             save_path=graph_filename,
175             overwrite=self.config.graph.overwrite,
176         )
```

Training Configuration

anemoi-core / training / src / anemoi / training / **config** /



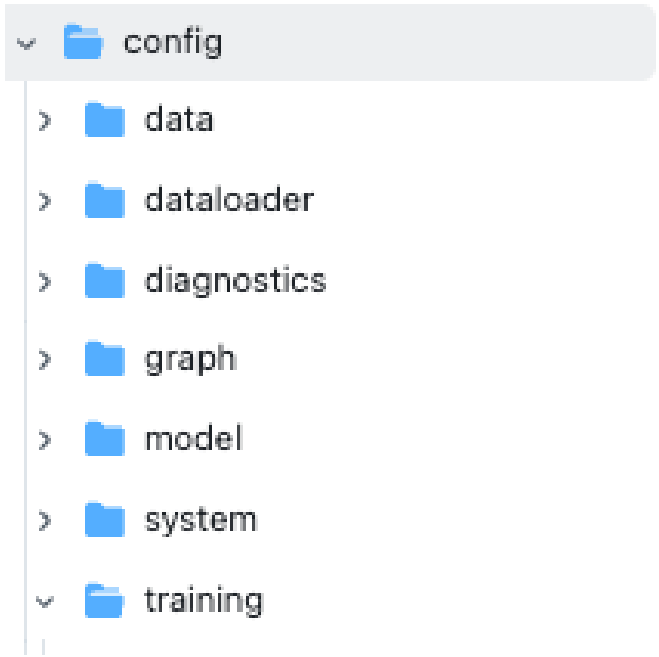

```yaml
defaults:
- data: zarr
- dataloader: native_grid
- diagnostics: evaluation
- system: example
- graph: multi_scale
- model: gnn
- training: default
- _self_


# 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

layer_kernels:
  # The layer_kernels can be adjusted per model component, but are d
  LayerNorm:
    # The GNN requires the autocast layer norm, otherwise its memory
    _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
```

- Set up to enable modifying key components of the models and training without code changes
- Using the Hydra config system
- Templates for various use cases based on a hierarchy of yaml files

ECMWF

Configs



data: These fields define the variables used by the model and how to normalise them

dataloader: These fields define parameters about we load the data specified in "data/".

diagnostics: Diagnostics produced during training with the validation dataset.

system: HPC/device specific configurations

graph: Your graph recipe

model: The graph neural network configuration

training: Training hyperparameters (learning_rate, loss function, ...)

Command Line Interface

- Generate user config folder:

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

- Start a training run with the *default configurations*:

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

- Start a training run with user settings:

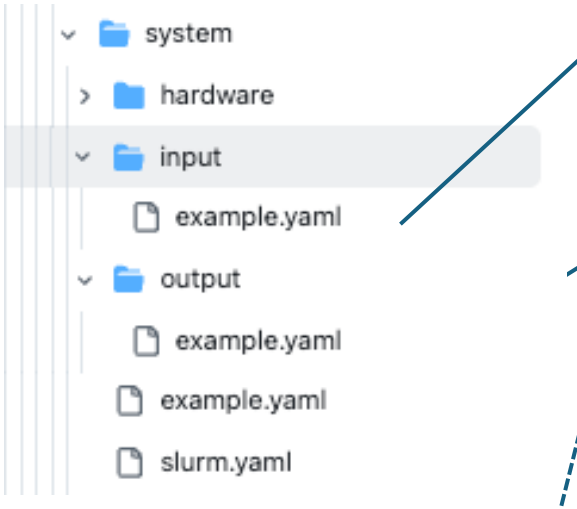
```
>>> anemoi-training train --config-name=my_config
```

Three ways to modify config entries:
- Edit directly in the copied config folder
- Add overrides in the top-level config
- Use cli overrides

The training script will intentionally crash as it does not know where your data is stored.

These missing values in the configuration are placeholders for the user to fill in marked with **???**.

How to point to where your data is stored




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

```yaml
root: ???
logs:
  root: logs
  wandb: wandb
  mlflow: mlflow
  tensorboard: tensorboard
checkpoints:
  root: checkpoint
  every_n_epochs: anemoi-by_epoch-epoch_{epoch:03d}-step_{step:06d}
  every_n_train_steps: anemoi-by_step-epoch_{epoch:03d}-step_{step:06d}
  every_n_minutes: anemoi-by_time-epoch_{epoch:03d}-step_{step:06d}
plots: plots
profiler: profiler
```

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
# Modifications for the multi dataset template "multi.yaml"
system:
  input:
    dataset: anemoi-integration-tests/training/datasets/aifs-ea-an-oper-0001-mars-o96-2017-2017-6h-v8-testing.zarr
    dataset_b: anemoi-integration-tests/training/datasets/cerra-rr-an-oper-0001-mars-5p5km-2017-2017-6h-v3-testing.zarr
```

Option 3 (as cli override)

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

ECMWF

Loss function

```yaml
# loss function for the model
training_loss:
  # loss class to initialise
  _target_: anemoi.training.losses.MSELoss
```

Deterministic Loss Functions

By default anemoi-training trains the model using a mean-squared-error, which is defined in the `MSELoss` class in `anemoi/training/losses/mse.py`. The loss function can be configured in the config file at `config.training.training_loss`, and `config.training.validation_metrics`.

The following loss functions are available by default:

- `MSELoss`: mean-squared-error.
- `RMSELoss`: root mean-squared-error.
- `MAELoss`: mean-absolute-error.
- `HuberLoss`: Huber loss.
- `LogCoshLoss`: log-cosh loss.
- `CombinedLoss`: Combined component weighted loss.

Probabilistic Loss Functions

The following probabilistic loss functions are available by default:

- `KernelCRPSLoss`: Kernel CRPS loss.
- `AlmostFairKernelCRPSLoss`: Almost fair Kernel CRPS loss see Lang et al. (2024).
- `WeightedMSELoss`: is the MSELoss used for the diffussion model to handle noise weights

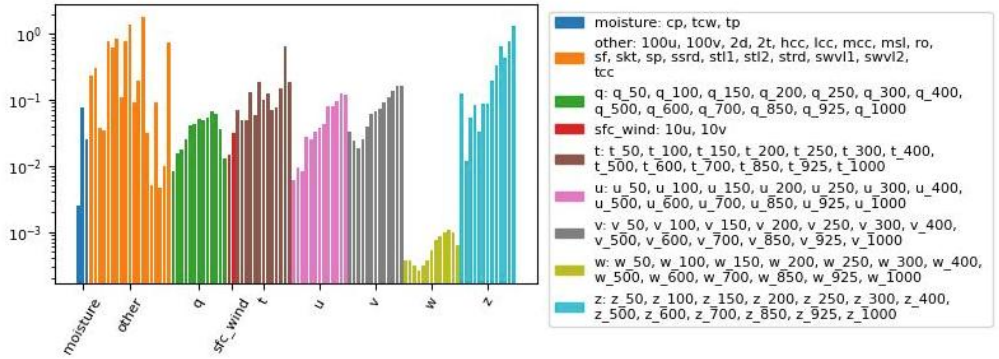
The config for these loss functions is the same as for the deterministic:

```yaml
# loss function for the model
training_loss:
   datasets:
      your_dataset_name:
         # loss class to initialise
         _target_: anemoi.training.losses.kcrps.KernelCRPSLoss
         # loss function kwargs here
```

Loss function

```yaml
# loss function for the model
training_loss:
  # loss class to initialise
  _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']
  ignore_nans: False
```

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



ECMWF

Scalers

```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
```

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

```yaml
node_weights:
  _target_: anemoi.training.losses.scalers.GraphNodeAttributeScaler
  nodes_name: ${graph.data}
  nodes_attribute_name: area_weight
  norm: unit-sum
```

Tasks

trainer

- `BaseGraphModule`: The abstract base class for all task-specific models, encapsulating shared logic for training, evaluation, and distributed execution.
- **Tasks**: Task-specific subclasses that implement models for deterministic forecasting, interpolation, ensemble learning, etc.
- `AnemoiTrainer`: The training orchestrator responsible for running and managing the training and validation loops.

To train a model, users typically subclass one of the pre-implemented graph modules or create a new one by extending `BaseGraphModule`.

Config (training section)

Graph (HeteroData)

Statistics

Data Indices

Supporting Arrays

Model (Lightning Module)

1. **Deterministic Forecasting** — GraphForecaster
2. **Ensemble Forecasting** — GraphEnsForecaster
3. **Time Interpolation** — GraphMultiOutInterpolator
4. **AutoEncoder** — GraphAutoEncoder

https://anemoi.readthedocs.io/projects/training/en/latest/modules/train.html#available-tasks

Tasks

trainer

↓

Model (Lightning Module) = Task

↓

Model (Torch Module) = Model

anemoi-core / training / src / anemoi / training / train / train.py

Code | Blame | 596 lines (499 loc) · 25.1 KB

```
                msg = f"Dataset source is None for dataset '{dataset_name}'. Check dataloader.dataset_config.dataset."
                raise ValueError(msg)
            graphs[dataset_name] = self._create_graph_for_dataset(dataset_source, dataset_name)
        return graphs

    @cached_property
    def model(self) -> pl.LightningModule:
        """Provide the model instance."""
        assert (
            not (
                "GLU" in self.config.model.processor.layer_kernels["Activation"]["_target_"]
                and ".Transformer" in self.config.model.processor.target_
            )
            and not (
                "GLU" in self.config.model.encoder.layer_kernels["Activation"]["_target_"]
                and ".Transformer" in self.config.model.encoder.target_
            )
            and not (
                "GLU" in self.config.model.decoder.layer_kernels["Activation"]["_target_"]
                and ".Transformer" in self.config.model.decoder.target_
            )
        ), "GLU activation function is not supported in Transformer models, due to fixed dimensions. "
        "Please use a different activation function."

        kwargs = {
            "config": self.config,
            "data_indices": self.data_indices,
            "graph_data": self.graph_data,
            "metadata": self.metadata,
            "statistics": self.datamodule.statistics,
            "statistics_tendencies": self.datamodule.statistics_tendencies,
            "supporting_arrays": self.supporting_arrays,
        }

        model_task = get_class(self.config.training.model_task)
        model = model_task(**kwargs)  # GraphForecaster -> pl.LightningModule

```

```
class BaseGraphModule(pl.LightningModule, ABC):
    def __init__(
        super().__init__()

        assert isinstance(graph_data, dict), "graph_data must be a dict keyed by dataset name"
        assert isinstance(data_indices, dict), "data_indices must be a dict keyed by dataset name"

        # Handle dictionary of graph_data
        graph_data = {name: data.to(self.device) for name, data in graph_data.items()}
        self.dataset_names = list(graph_data.keys())

        # Create output_mask dictionary for each dataset
        self.output_mask = {}
        for name in self.dataset_names:
            self.output_mask[name] = instantiate(config.model.output_mask, graph_data=graph_data[name])

        # Handle supporting_arrays merge with all output masks
        combined_supporting_arrays = supporting_arrays.copy()
        for dataset_name, mask in self.output_mask.items():
            combined_supporting_arrays[dataset_name].update(mask.supporting_arrays)

        if not hasattr(self.__class__, "task_type"):
            msg = """Subclasses of BaseGraphModule must define a `task_type` class attribute,
                indicating the type of task (e.g., 'forecaster', 'time-interpolator')."""
            raise AttributeError(msg)

        metadata["metadata_inference"]["task"] = self.task_type

        self.model = AnemoiModelInterface(
            statistics=statistics,
            statistics_tendencies=statistics_tendencies,
            data_indices=data_indices,
            metadata=metadata,
            supporting_arrays=combined_supporting_arrays,
            graph_data=graph_data,
            config=config,
        )
        self.config = config

        self.data_indices = data_indices

        self.save_hyperparameters()

        self.statistics_tendencies = statistics_tendencies
```

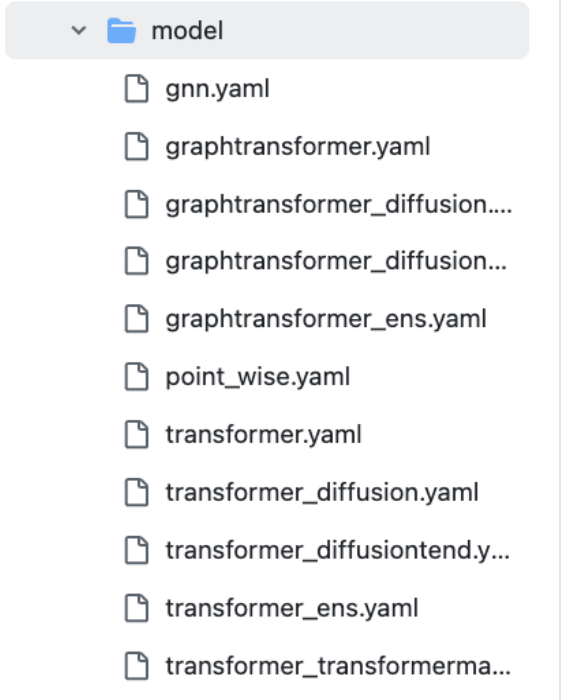
ECMWF

https://anemoi.readthedocs.io/projects/training/en/latest/modules/train.html#available-tasks

Configs

```
anemoi-training train model=gnn

anemoi-training train model=graphtransformer

anemoi-training train model=transformer
```

- Make it easy to switch components
- Allow for reproduceable training
- Easy to extend with new models and components




```yaml
num_channels: 1024
cpu_offload: False

keep_batch_sharded: True

model:
  _target_: anemoi.models.models.AnemoiModelEncProcDec

processor:
  _target_: anemoi.models.layers.processor.GraphTransformerProcessor
  trainable_size: ${model.trainable_parameters.hidden2hidden}
  sub_graph_edge_attributes: ${model.attributes.edges}
  num_layers: 16
  num_chunks: 4
  mlp_hidden_ratio: 4 # GraphTransformer or Transformer only
  num_heads: 16 # GraphTransformer or Transformer only
  qk_norm: False
  cpu_offload: ${model.cpu_offload}
  gradient_checkpointing: True
  layer_kernels: ${model.layer_kernels}
  graph_attention_backend: "triton"  # Options: "triton", "pyg"
  edge_pre_mlp: False

encoder:
  _target_: anemoi.models.layers.mapper.GraphTransformerForwardMapper
  trainable_size: ${model.trainable_parameters.data2hidden}
  sub_graph_edge_attributes: ${model.attributes.edges}
  num_chunks: 4
  mlp_hidden_ratio: 4 # GraphTransformer or Transformer only
  num_heads: 16 # GraphTransformer or Transformer only
  qk_norm: False
  cpu_offload: ${model.cpu_offload}
  gradient_checkpointing: True
  layer_kernels: ${model.layer_kernels}
  shard_strategy: "edges"
  graph_attention_backend: "triton"  # Options: "triton", "pyg"
  edge_pre_mlp: False

decoder:
  _target_: anemoi.models.layers.mapper.GraphTransformerBackwardMapper
  trainable_size: ${model.trainable_parameters.hidden2data}
  sub_graph_edge_attributes: ${model.attributes.edges}
  num_chunks: 4
  mlp_hidden_ratio: 4 # GraphTransformer or Transformer only
  num_heads: 16 # GraphTransformer or Transformer only
  initialise_data_extractor_zero: False
  qk_norm: False
  cpu_offload: ${model.cpu_offload}
  gradient_checkpointing: True
  layer_kernels: ${model.layer_kernels}
  shard_strategy: "edges"
  graph_attention_backend: "triton"  # Options: "triton", "pyg"
  edge_pre_mlp: False
```

Schemas

Bringing schema and sanity to your data

ECMWF

Config validation

- In anemoi training, runs are **controlled** by **YAML** configuration files defining datasets, model architecture, hyperparameters, ...
- Misconfigured fields can silently break runs or produce inconsistent results.
- Schema validation ensures:
  - **Consistency**: all configs follow the same structure and naming.
  - **Error prevention**: catches typos, missing fields, wrong data 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
```

Pydantic

Bringing schema and sanity to your data

ECMWF

Schemas

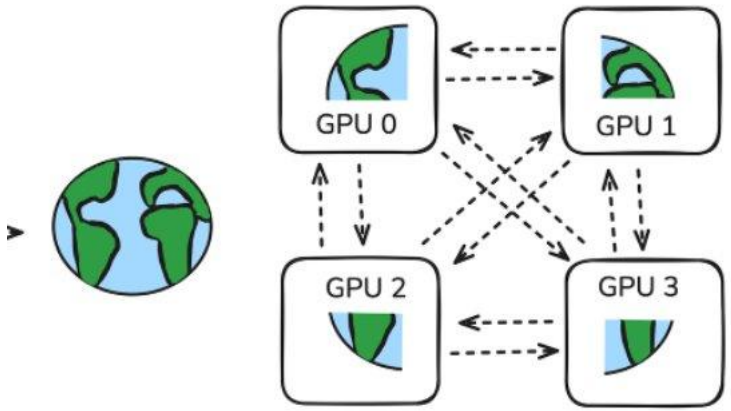
```yaml
defaults:
- data: zarr
- dataloader: native_grid
- diagnostics: evaluation
- system: example
- graph: multi_scale
- model: gnn
- training: default
- _self_


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

```python
class DatasetDataSchema(PydanticBaseModel):
    """A class used to represent the configuration of a single dataset."""

    forcing: list[str] = Field(default_factory=list)
    "Features that are not part of the forecast state but are used as forcing to generate the forecast state."
    diagnostic: list[str] = Field(default_factory=list)
    "Features that are only part of the forecast state and are not used as an input to the model."
    target: list[str] | None = None
    (
        "Features used to compute the loss against forecasted variables. "
        "Cannot be prognostic or diagnostic, can have the same name as forcing variables "
        "but have a different role. Such that: prognostic = diagnostic - forcing.union(target)."
    )

    processors: dict[str, PreprocessorSchema]
    "Layers of model performing computation on latent space. \
        Processors including imputers and normalizers are applied in order of definition. (single dataset mode)"
```

```python
class DataSchema(PydanticBaseModel):
    """A class used to represent the overall configuration of the dataset(s).

    Attributes
    ----------
    format : str
        The format of the data.
    resolution : str
        The resolution of the data.
    frequency : str
        The frequency of the data.
    timestep : str
        The timestep of the data.
    datasets : dict[str, DatasetDataSchema] | None
        "Dictionary mapping dataset names to their configurations."
    num_features : int, optional
        The number of features in the forecast state. To be set in the code.
    """

    format: str = Field(example=None)
    "Format of the data."
    frequency: str = Field(example=None)
    "Time frequency requested from the dataset."
    timestep: str = Field(example=None)
    "Time step of model (must be multiple of frequency)."
    datasets: dict[str, DatasetDataSchema] | None = None
    "Dictionary mapping dataset names to their configurations."
    num_features: int | None
    "Number of features in the forecast state. To be set in the code."
```

ECMWF

Scaling your models



ECMWF

Why do we parallelise?

anemoi-core / training / src / anemoi / training / config / training / **default.yaml**

Code | Blame | 197 lines (168 loc) · 7.01 KB

```yaml
    # ------------------------------------------------------------------
    # Optional: configuration for AdEMAMix (custom optimizer)
    # Uncomment the lines below to enable it
    # ------------------------------------------------------------------
    # _target_: anemoi.training.optimizers.AdEMAMix.AdEMAMix  # Custom optimizer
    # betas: [0.9, 0.95, 0.9999]  # β₁, β₂, β₃
    # alpha: 8.0                  # Mixing factor controlling EMA fusion
    # beta3_warmup: 260000        # Warm-up steps for β₃ (in iterations)
    # alpha_warmup: 260000        # Warm-up steps for α (in iterations)
    # weight_decay: 0.01

    # Optional: configuration for ZeroRedundancyOptimizer
    # _target_: torch.distributed.optim.ZeroRedundancyOptimizer
    # optimizer_class:
    #   _target_: torch.optim.AdamW
    #   _partial_: true
    # betas: [0.9, 0.95]

  # select model
  model_task: anemoi.training.train.tasks.GraphForecaster

  # 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}

```

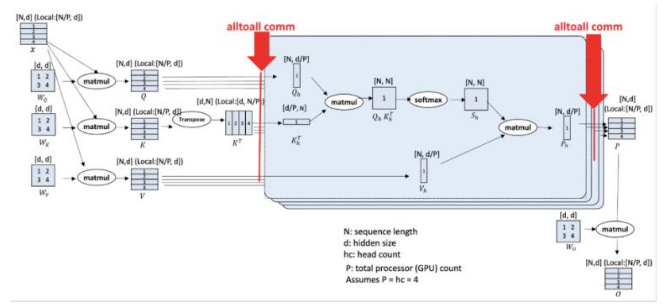
Data-Distributed

This is used automatially if the `number of parallel data GPUs = number of GPUs available/number of GPUs per model` is an integer greater than 1. In this case the batches will be split across the number of parallel data GPUs meaning that the effective batch size of each training step will be the number of batches set in the dataloader config file multiplied by the number of parallel data GPUs.

Model Sharding

It is also possible to shard the model across multiple GPUs as defined by the Distributed Data Parallel (DDP) Strategy.

In essence the model is sharded with each GPU receiving a different part of the graph, before being re-integrated when the loss is calculated, as shown in the figure below



Model Sharding (source: Jacobs et al. (2023))

To use model sharding, set `config.system.hardware.num_gpus_per_model` to the number of GPUs you wish to shard the model across. Set `config.model. keep_batch_sharded=True` to also keep batches fully sharded throughout training, reducing memory usage for large inputs or long rollouts. It is recommended to only shard if the model does not fit in GPU memory, as data distribution is a much more efficient way to parallelise the training.

Why do we parallelise?

anemoi-core / training / src / anemoi / training / config / model / graphtransformer.yaml

5 people feat(training): make activation checkpoints configurable (#797)

Code Blame 146 lines (129 loc) · 5.29 KB

```yaml
num_channels: 1024
cpu_offload: False

keep_batch_sharded: True

model:
  _target_: anemoi.models.models.AnemoiModelEncProcDec

layer_kernels:
  # The layer_kernels can be adjusted per model component, but are defined here for convenience.
  LayerNorm:
    _target_: torch.nn.LayerNorm
  Linear:
    _target_: torch.nn.Linear
  Activation:
    _target_: torch.nn.GELU
  QueryNorm:
    _target_: anemoi.models.layers.normalization.AutocastLayerNorm
    bias: False
  KeyNorm:
    _target_: anemoi.models.layers.normalization.AutocastLayerNorm
    bias: False

processor:
  _target_: anemoi.models.layers.processor.GraphTransformerProcessor
  trainable_size: ${model.trainable_parameters.hidden2hidden}
  sub_graph_edge_attributes: ${model.attributes.edges}
  num_layers: 16
  num_chunks: 4
  mlp_hidden_ratio: 4 # GraphTransformer or Transformer only
  num_heads: 16 # GraphTransformer or Transformer only
  qk_norm: False
  cpu_offload: ${model.cpu_offload}
  gradient_checkpointing: True
  layer_kernels: ${model.layer_kernels}
  graph_attention_backend: "triton"  # Options: "triton", "pyg"
  edge_pre_mlp: False

encoder:
  _target_: anemoi.models.layers.mapper.GraphTransformerForwardMapper
  trainable_size: ${model.trainable_parameters.data2hidden}
  sub_graph_edge_attributes: ${model.attributes.edges}
  num_chunks: 4
  mlp_hidden_ratio: 4 # GraphTransformer or Transformer only
  num_heads: 16 # GraphTransformer or Transformer only
  qk_norm: False
  cpu_offload: ${model.cpu_offload}
  gradient_checkpointing: True
  layer_kernels: ${model.layer_kernels}
  shard_strategy: "edges"
  graph_attention_backend: "triton"  # Options: "triton", "pyg"
  edge_pre_mlp: False

```

Shard model over more GPUs

If your model instance still does not fit in memory, you can shard the model over multiple GPUs.

```yaml
system:
   hardware:
      num_gpus_per_model: 2
```

This will reduce memory usage by sharding the input batch and model channels across GPUs.

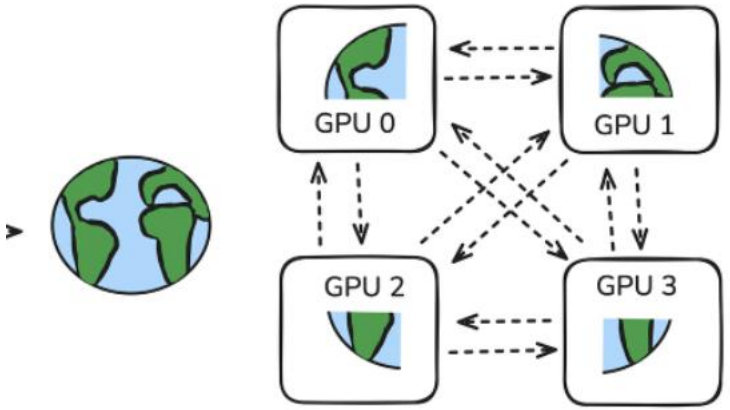
The number of GPUs per model should be a power of two and is limited by the number of heads in your model, by default 16.

Sharding a model over multiple GPUs can also increase performance, as the compute workload is divided over more GPUs. However sharding a model over too many GPUs can lead to decreased performance from increased collective communication operations required to keep the GPUs in sync. Additionally, model sharding increases the total number of GPUs required, which can lead to longer queue times on shared HPC systems.

The GPUs within a node typically are connected via a faster interconnect than GPUs across nodes. For this reason model sharding typically performs less efficiently once a model instance is sharded across multiple nodes.

Why do we parallelise?

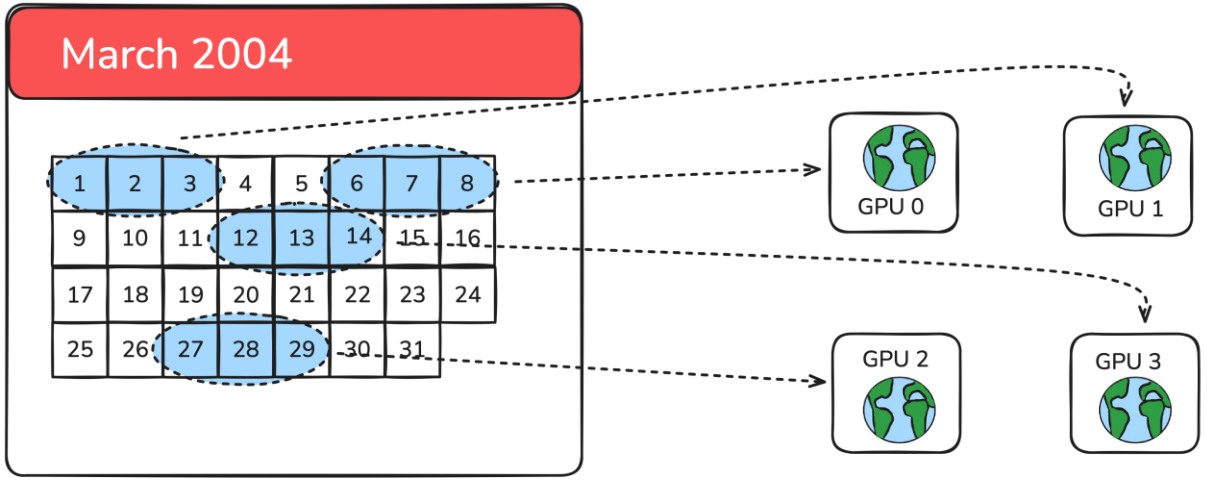
- Faster training!
  - Share the work across multiple GPUs e.g. 1 attention head per GPU
  - Each can compute a fraction in parallel
  - … and sychronise their results via messages at the end



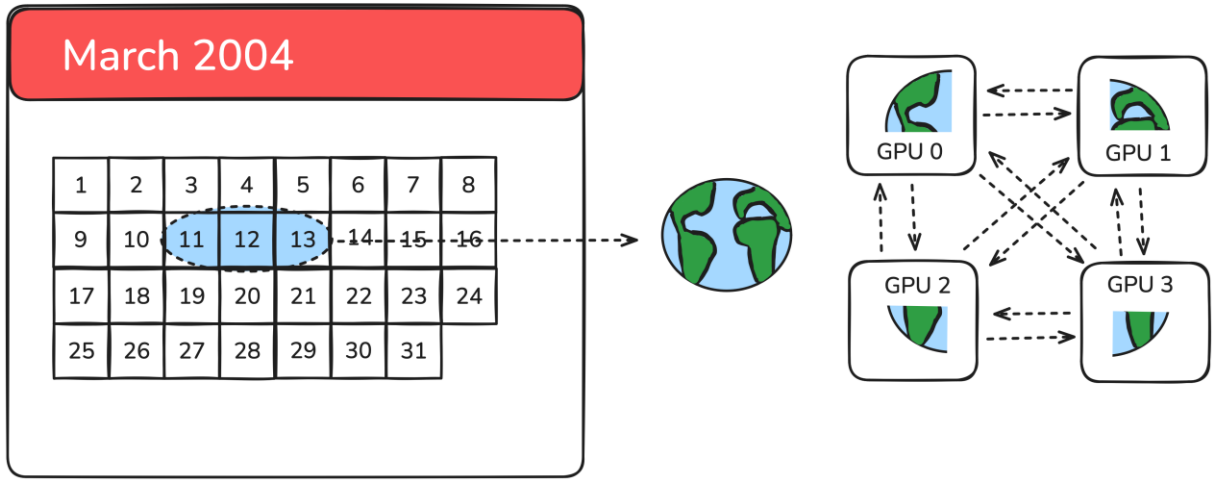
- Larger models!
  - Split the globe across multiple GPUs
  - Each GPU only needs to communicate the boundary conditions
  - Never materialise the full globe in a single GPUs memory

Parallelism in Anemoi

- **Data Parallelism:** DistributedDataParallel (PyTorch)

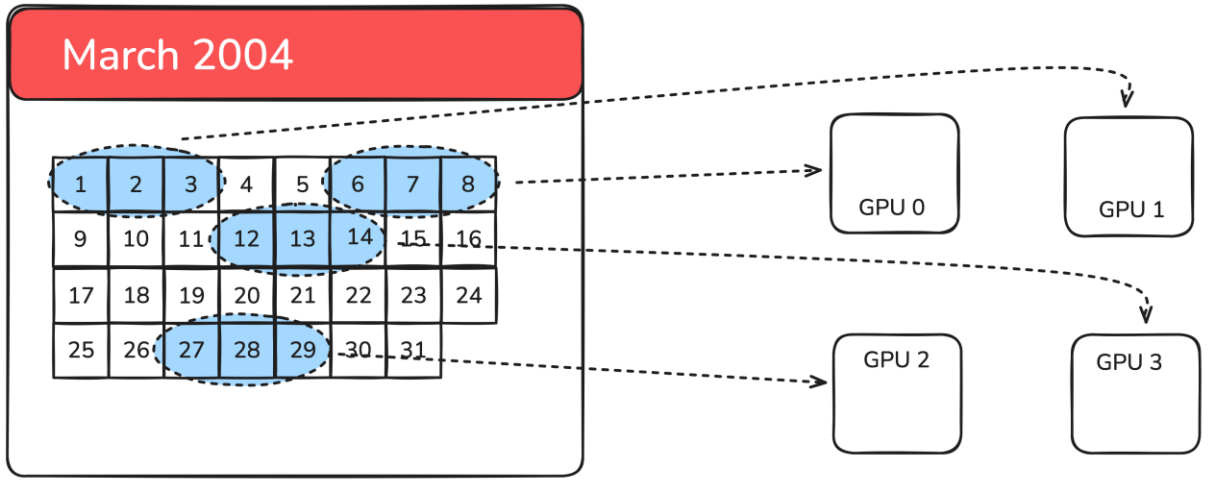


- **Model Parallelism:** domain-specific sharding

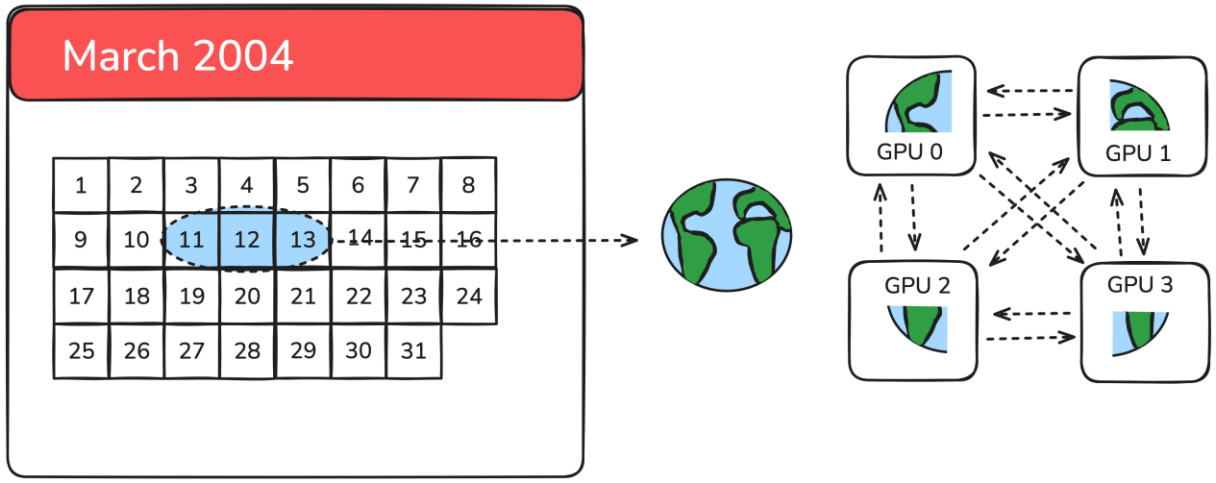


Parallelism in Anemoi

- Data Parallelism: DistributedDataParallel (PyTorch)
  - Distribute training batch across model replicas
  - Aggregate gradients via all-reduce, good scaling 🙂
  - Limited by batch size 😟

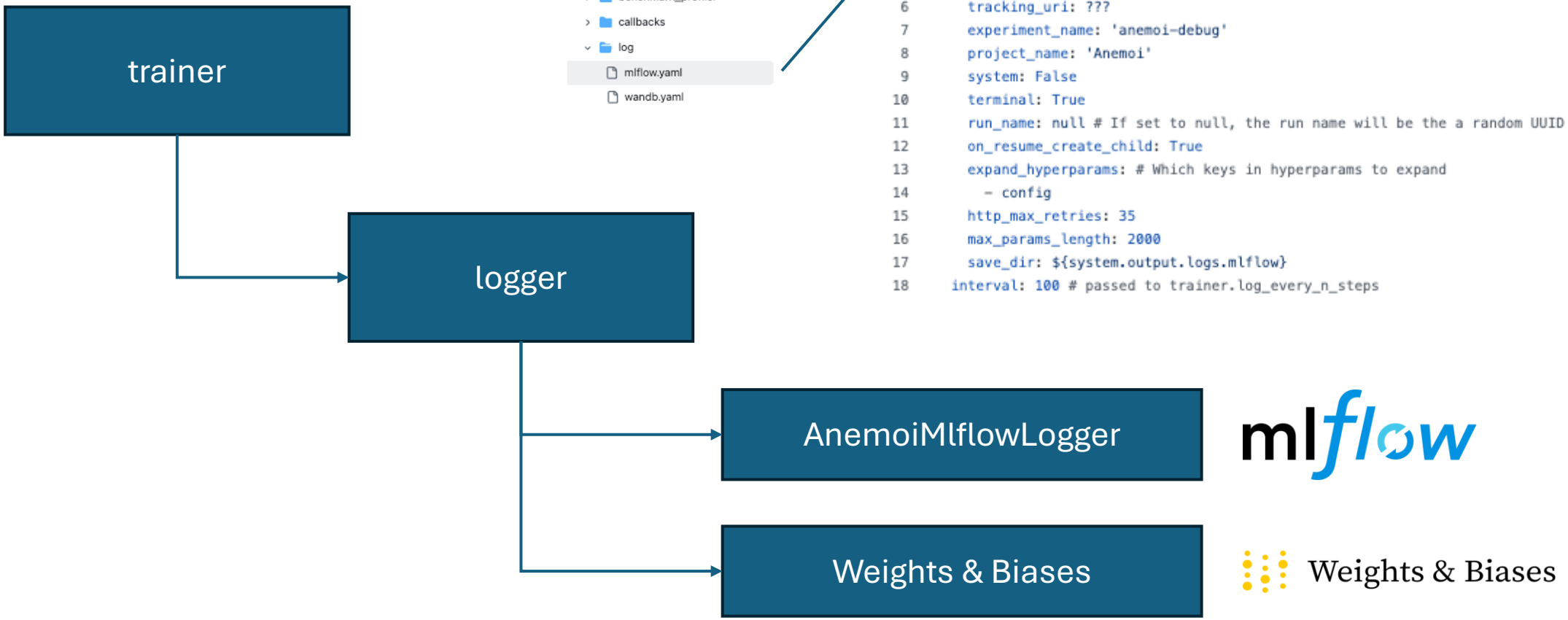


- Model parallelism: domain-specific sharding
  - Distribute input data and activations across GPUs
  - Collective communication to handle synchronisation
  - Limited by communication overheads

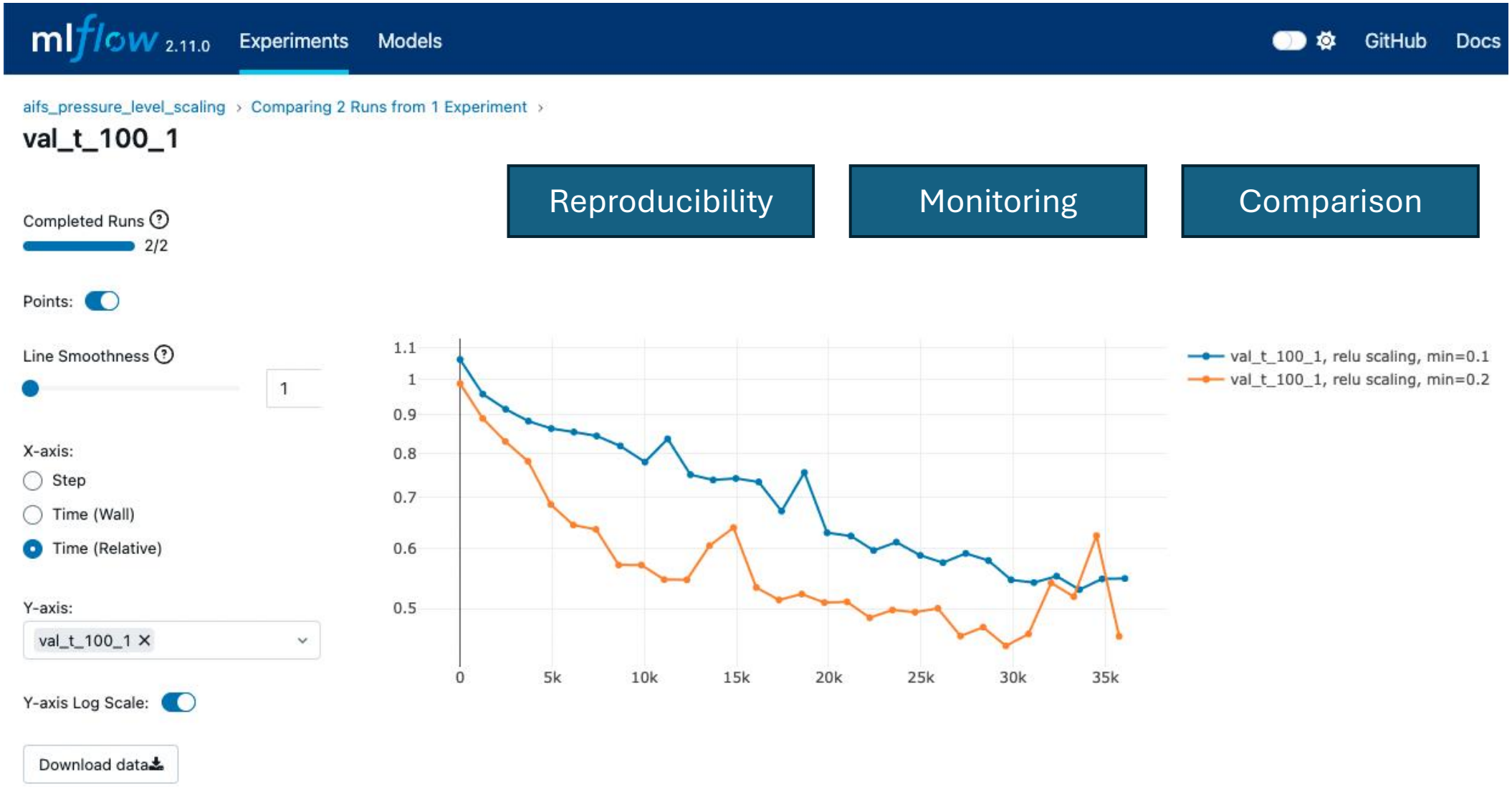


Experiment tracking

Experiment tracking

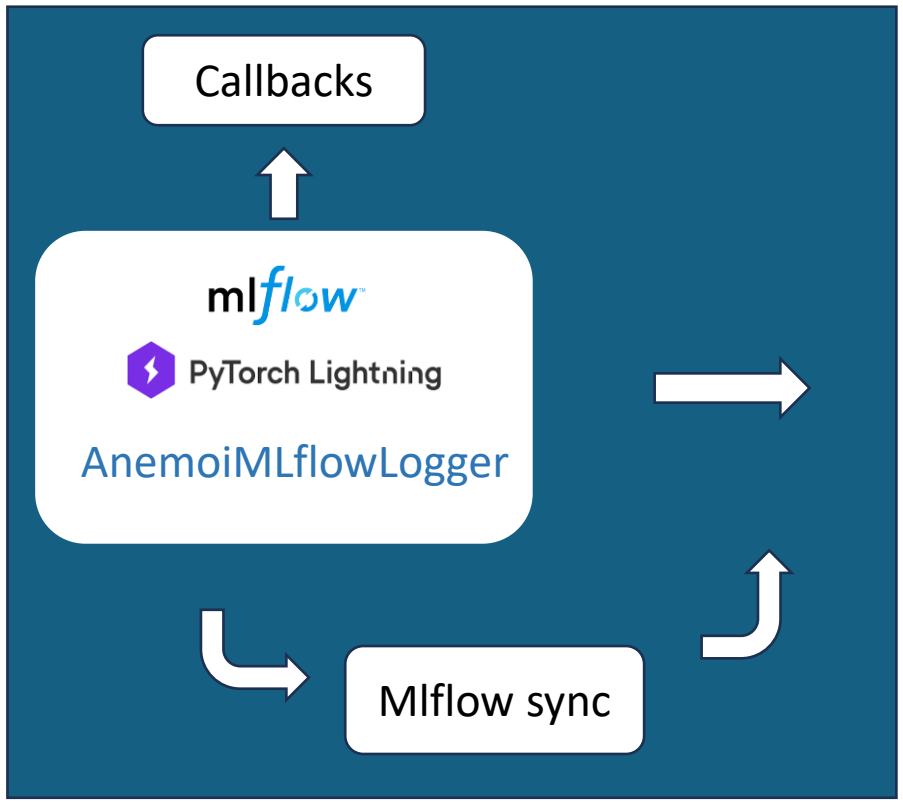


Anemoi.training – MLFlow servers for Experiment Tracking



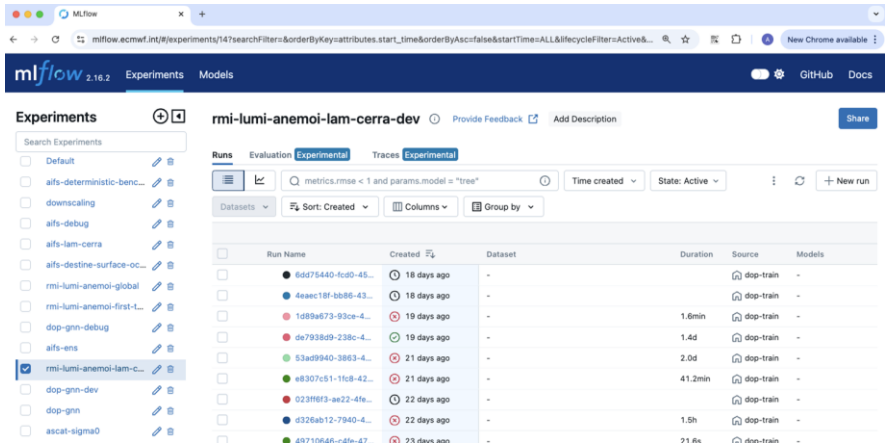
MLflow server

Anemoi-training



Internet

Console or web browser



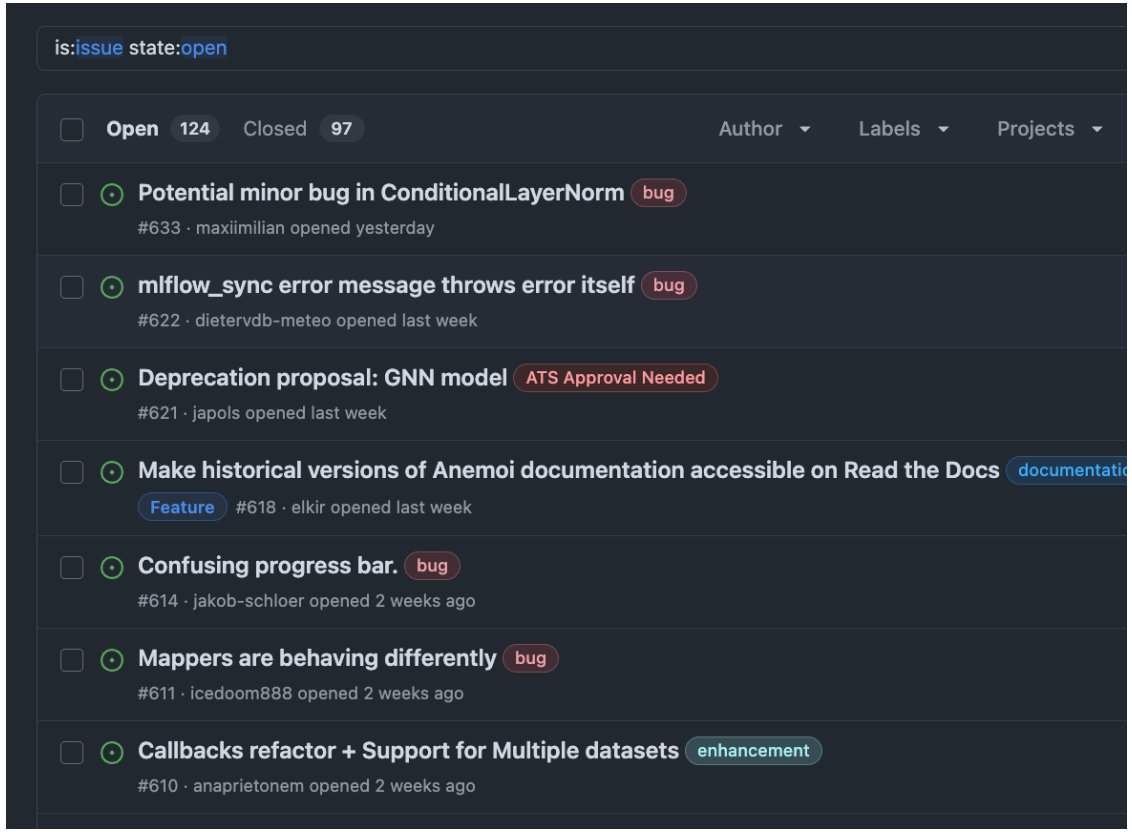
ECMWF

Contributing to anemoi

ECMWF

Open an Issue

- Go to the repository (e.g. `github.com/ecmwf/anemoi-core`).
  - Click **“Issues” → “New issue.”**
  - Choose the right template (bug report, feature request).
- Fill in clear details:
  - What happened?
  - What are the steps to reproduce the bug?
  - What happened (include logs, error messages)
  - Environment (Python version, OS, dependencies)
- **Submit:** maintainers will triage and respond.



https://anemoi.readthedocs.io/en/latest/contributing/contributing.html

ECMWF

Create a Pull Request (PR)

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

- Fork the repository to your GitHub account.
- Clone your fork locally and create a new branch:

```bash
git checkout -b fix-dataloader-bug
```

- Make and test your changes.
- Commit with a clear message.
- Push your branch and open a PR on GitHub.
  - In the PR description, explain what you changed and *why*.
- Wait for automated checks & maintainer review.

Additionally, contributors can tag `@ecmwf/anemoitechnicalsubgroup` in issues or PRs to explicitly highlight topics requiring subgroup attention. This can be especially helpful to summarize a disagreement, flag a potentially controversial feature, or raise visibility for decisions that require alignment across institutions.

Anemoi Security Group

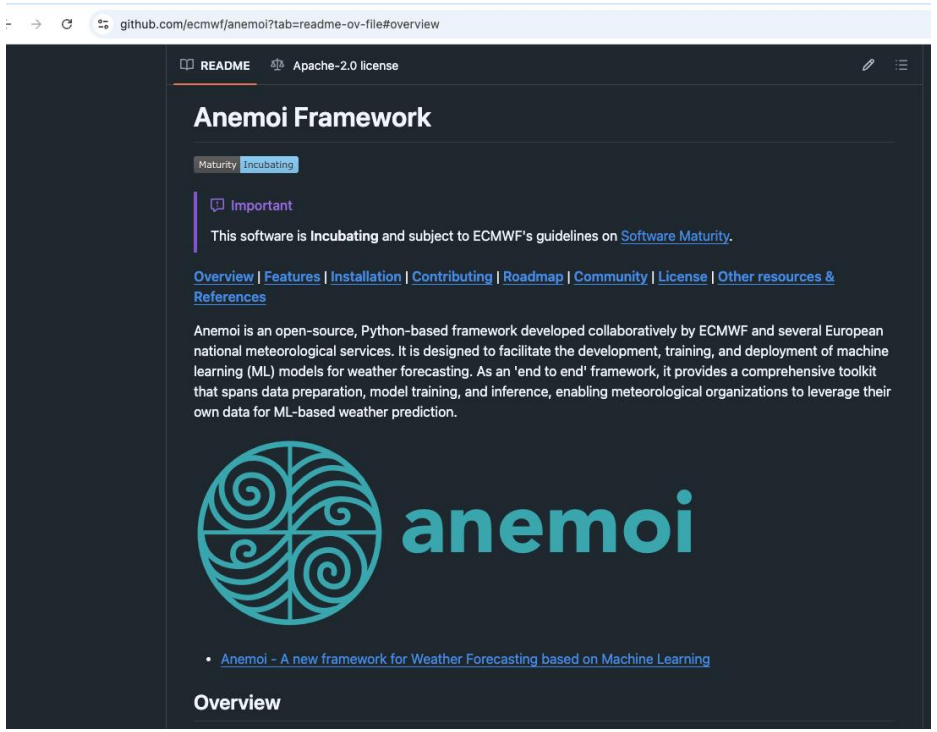
The `@ecmwf/anemoisecurity` group is a core part of Anemoi's governance and review process. Members of this group are automatically tagged in pull requests that touch sensitive or protected areas of the codebase, as defined in each repository's `CODEOWNERS` file. Their role is to ensure consistent application of best practices, integrity, and to offer early guidance where there may be ambiguity or technical disagreement—before discussion at the Anemoi Technical Subgroup (ATS).

https://anemoi.readthedocs.io/en/latest/contributing/governance.html

ECMWF

Keep Learning about anemoi-training

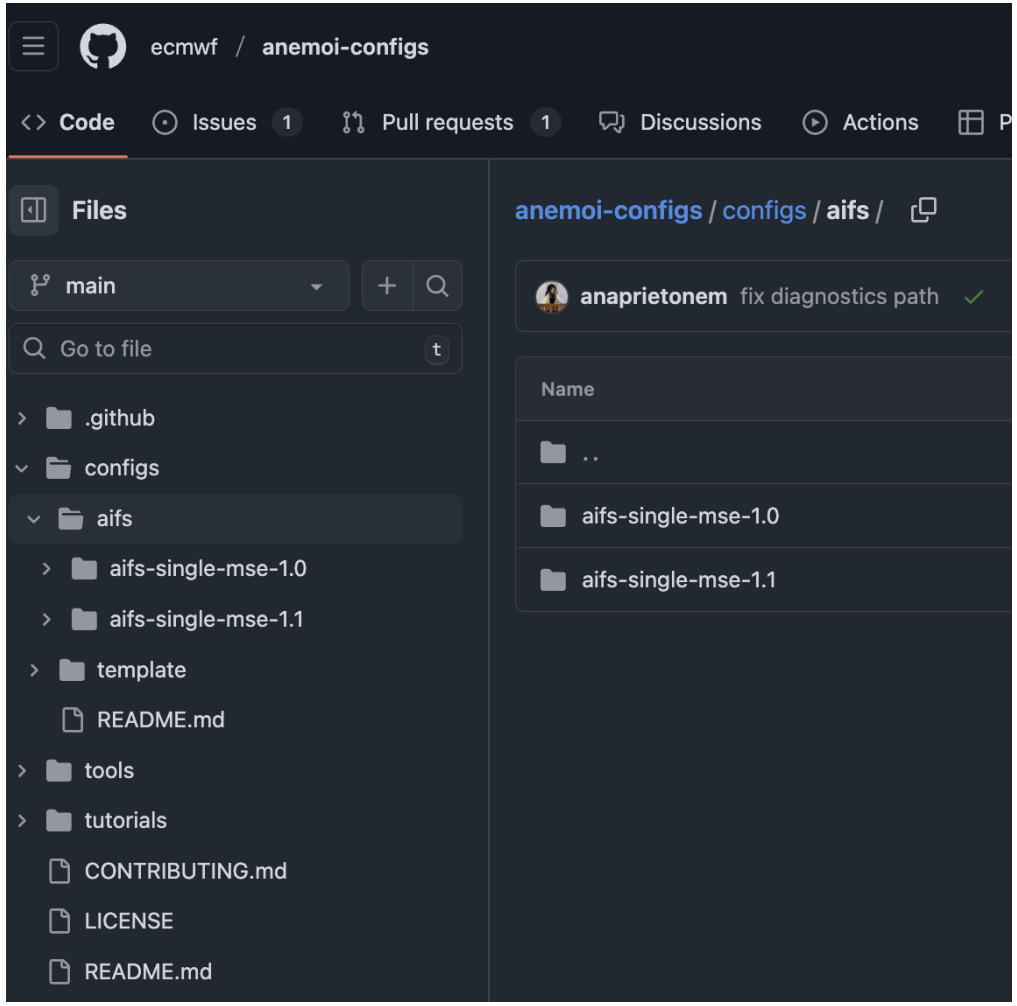
- Docs: https://anemoi.readthedocs.io/projects/training/en/latest/
- Config templates: https://github.com/ecmwf/anemoi-core/tree/main/training/src/anemoi/training/config
- Integration tests configs: https://github.com/ecmwf/anemoi-core/tree/main/training/tests/integration

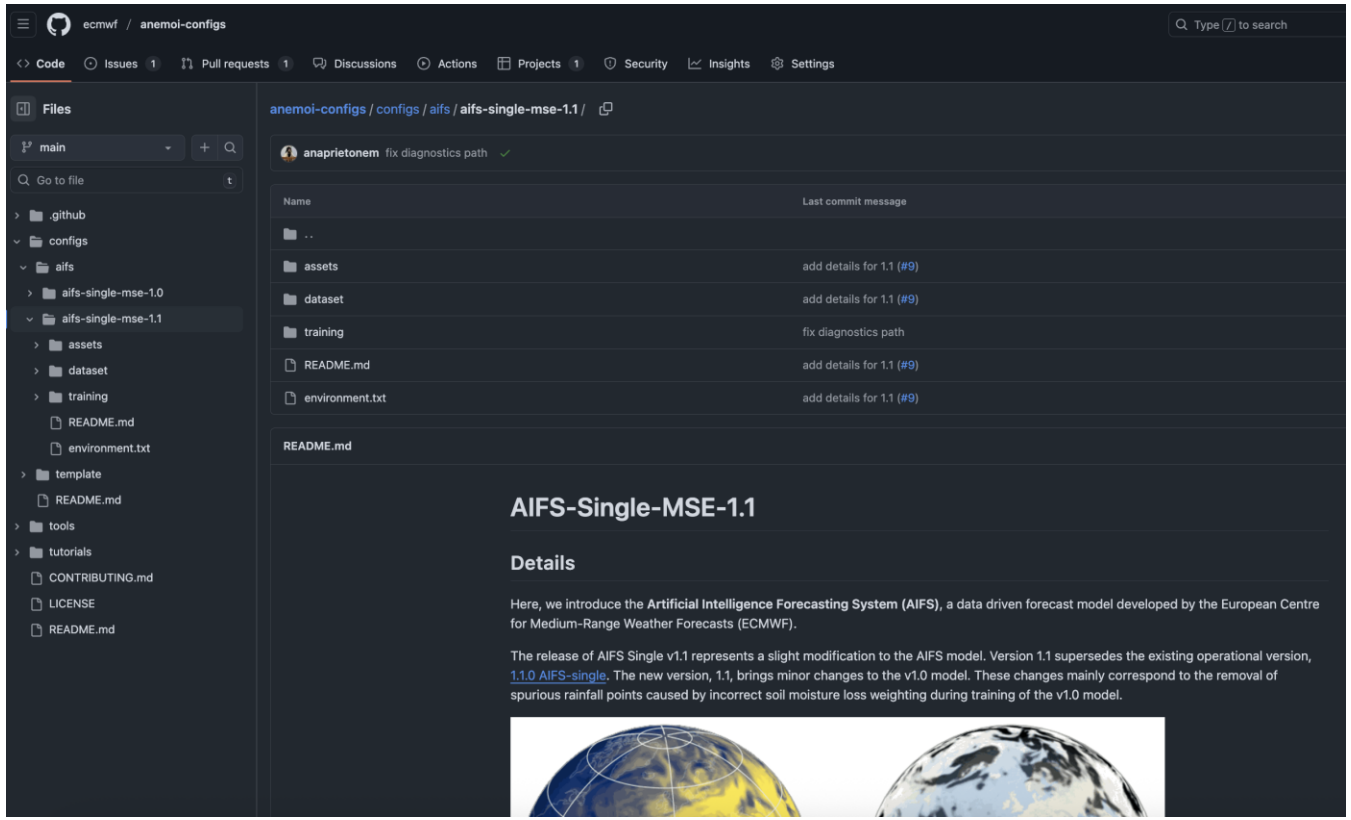


- https://github.com/ecmwf/anemoi

ECMWF

anemoi-configs





ECMWF

Questions