Anemoi for Developers

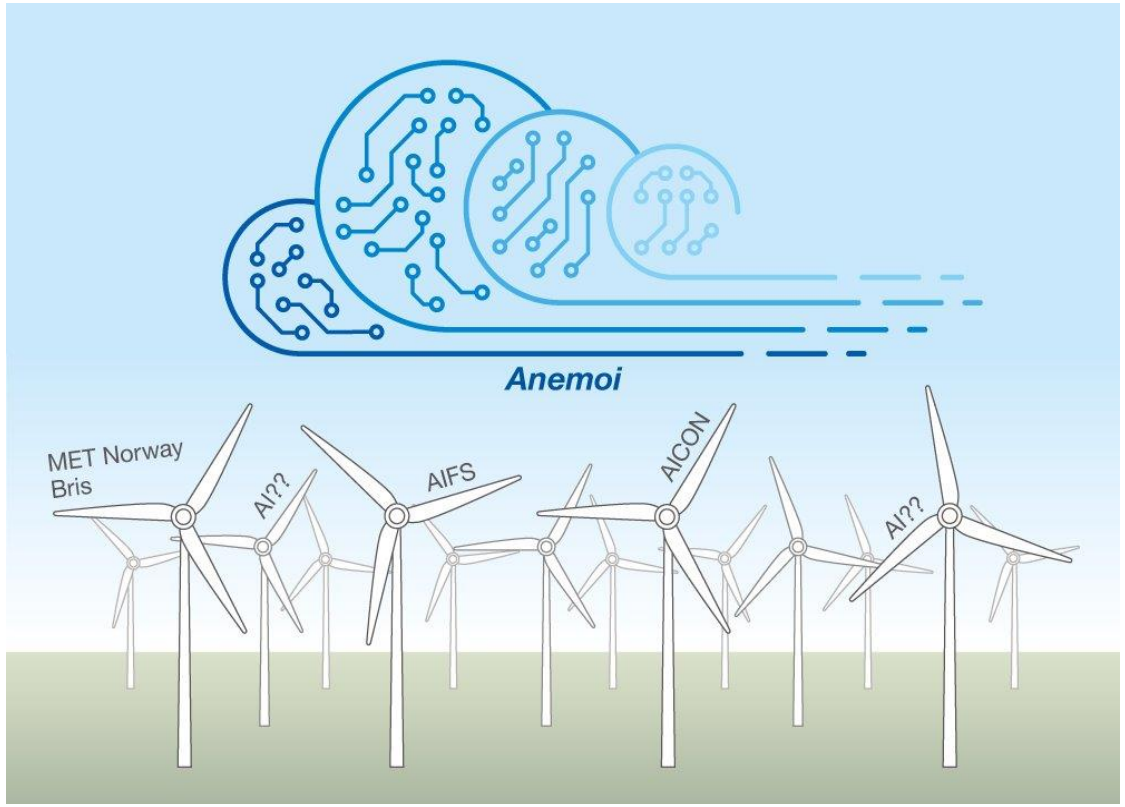
Helen Theissen

ECMWF

helen.theissen@ecmwf.int

Anemoi framework

- Developing machine learning (ML) weather forecasting models.



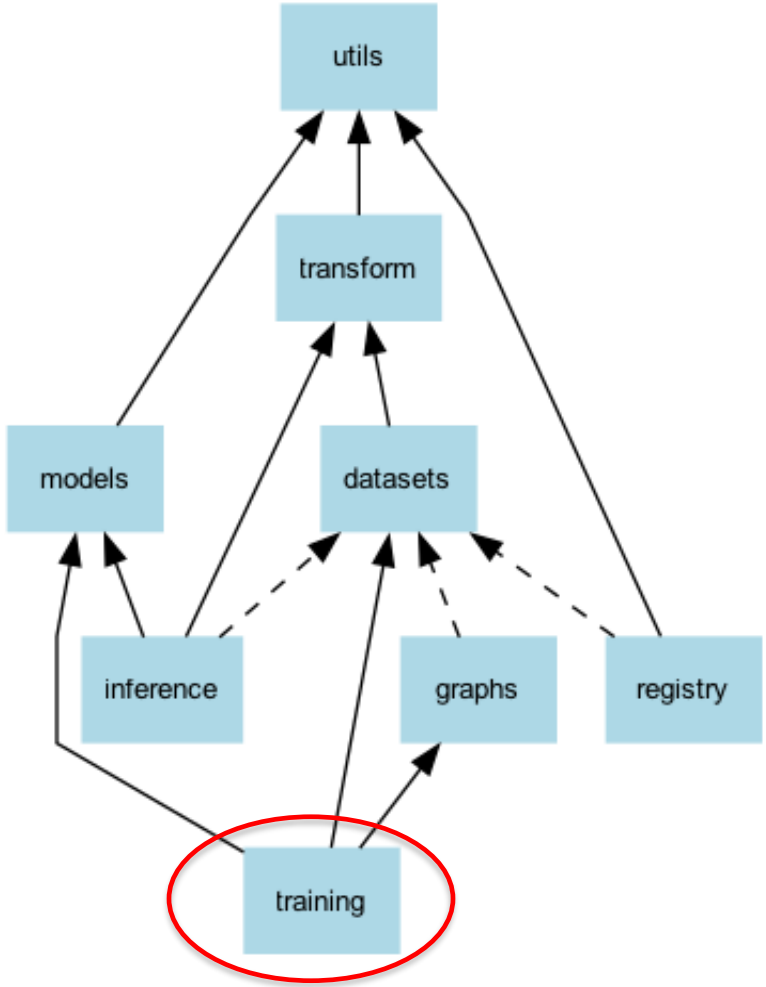
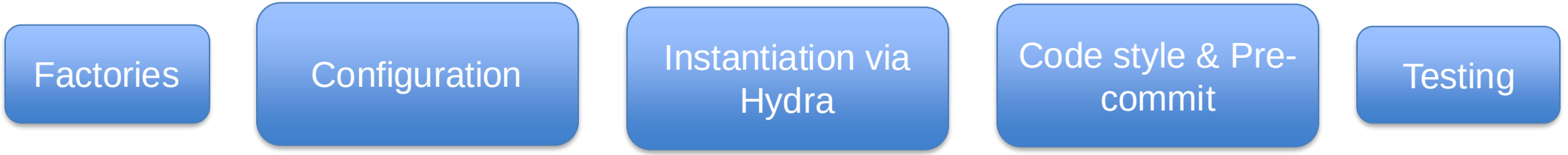


Diagram of dependencies within anemoi packages.

From Idea to Realization

- Integrate a new loss function and open a PR in anemoi-core
- **Mean Squared Logarithmic Error (MSLE)**

$$\frac{1}{N}\sum_{i=0}^{N}(\log(y_i+1)-\log(\hat{y}_i+1))^2$$



How do I start developing?

1. Clone anemoi-core

```
$ git clone https://github.com/ecmwf/anemoi-core.git
$ cd anemoi-core
```

2. Create environment

```
$ conda create -n anemoi-env python=3.12
$ conda activate anemoi-env
```

3. Install packages

```
$ pip install -e ./graphs[all, tests]
$ pip install -e ./models[all, tests]
$ pip install -e ./training[all, tests]
```

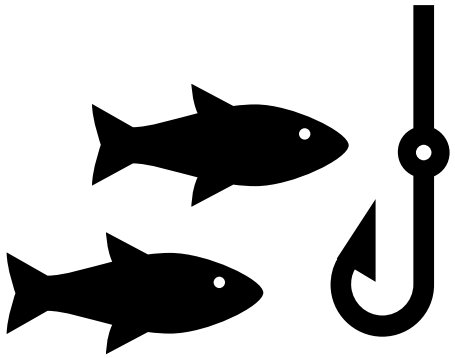
4. Install pre-commit hooks

```
$ pip install pre-commit
$ pre-commit install
```

Pre-commit

- Pre-commit hooks run before the actual commit
- Prevents submitting unformatted code
- Formatting code (e.g. auto-formatting with black)
- Linting code (e.g. check for style violations)

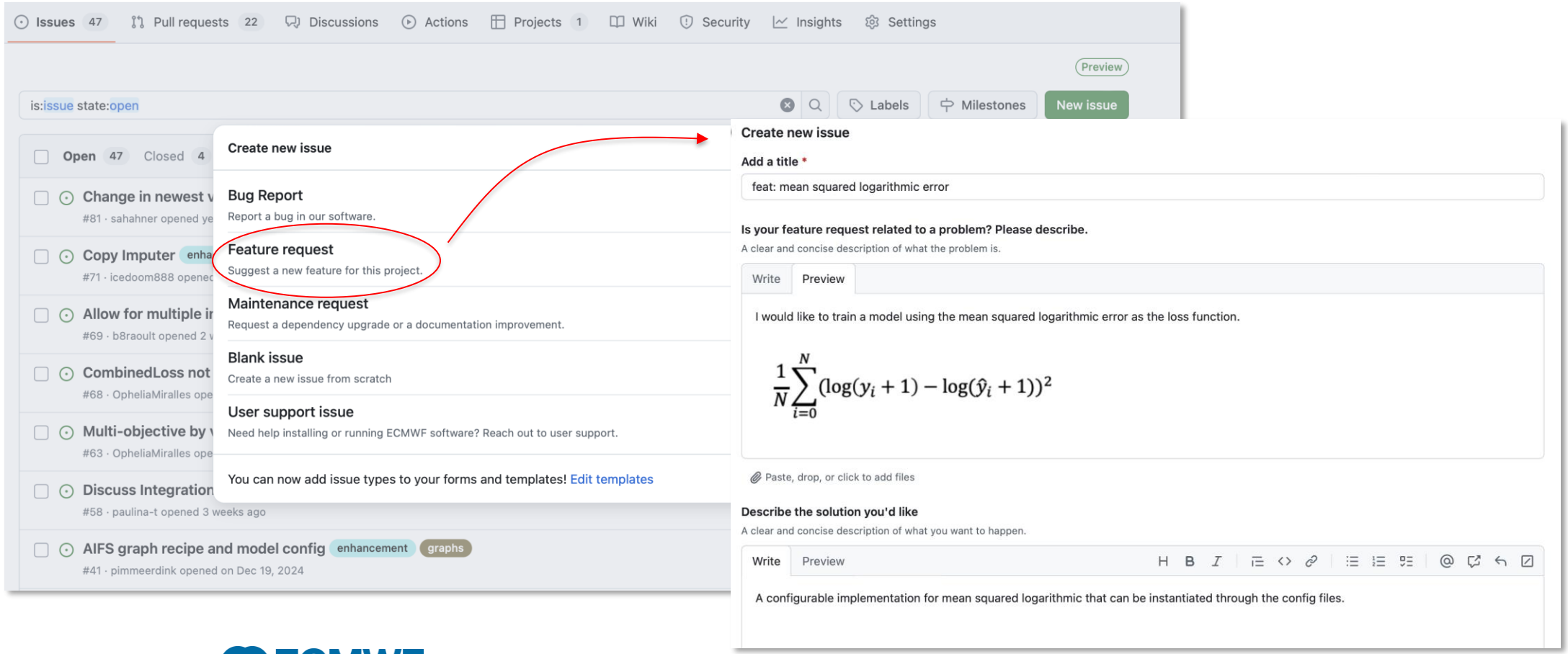
```yaml
repos:      Baudouin Raoult, 7 months ago • First commit
# Empty notebookds
- repo: local
  hooks:
  - id: clear-notebooks-output
    name: clear-notebooks-output
    files: tools/.*\.ipynb$
    stages: [pre-commit]
    language: python
    entry: jupyter nbconvert --ClearOutputPreprocessor.enabled=True --inplace
    additional_dependencies: [jupyter]
- repo: https://github.com/pre-commit/pre-commit-hooks
  rev: v5.0.0
  hooks:
  - id: check-yaml # Check YAML files for syntax errors only
    args: [--unsafe, --allow-multiple-documents]
  - id: debug-statements # Check for debugger imports and py37+ breakpoint()
  - id: end-of-file-fixer # Ensure files end in a newline
  - id: trailing-whitespace # Trailing whitespace checker
  - id: no-commit-to-branch # Prevent committing to main / master
  - id: check-added-large-files # Check for large files added to git
  - id: check-merge-conflict # Check for files that contain merge conflict
```



.pre-commit-config.yaml

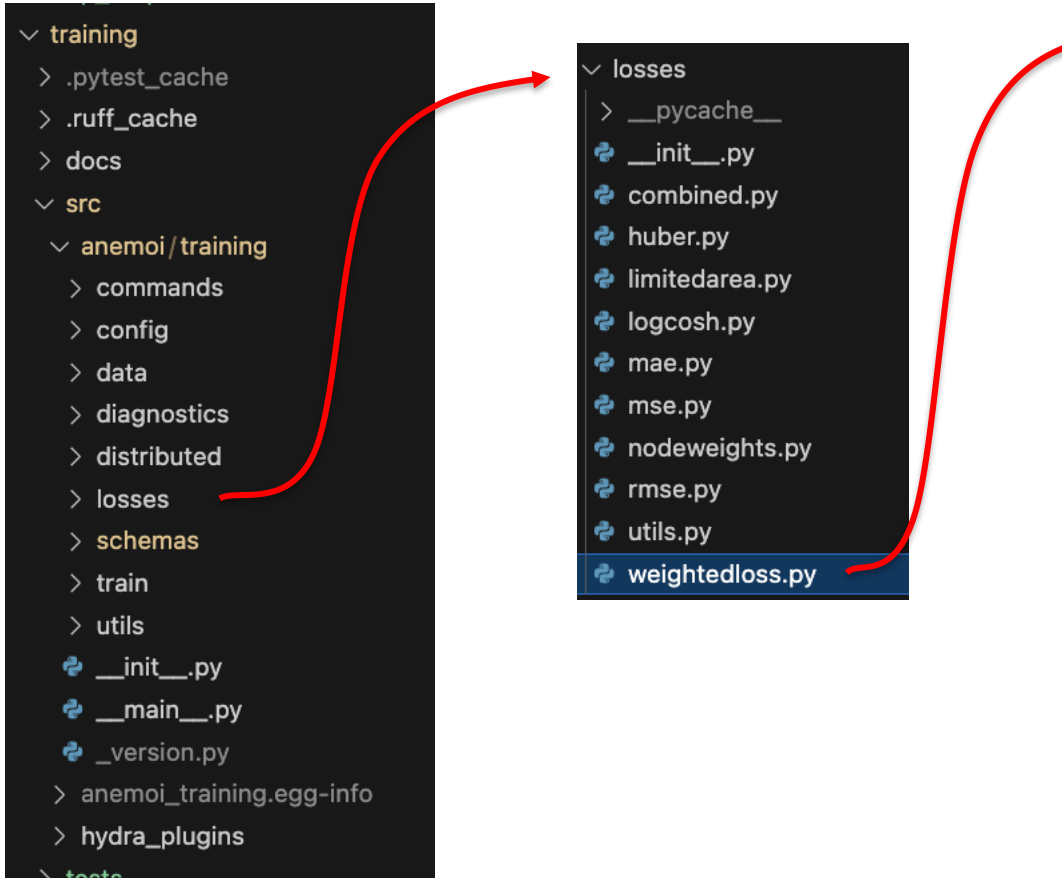
Creating an issue and branch

- Create issue/branch
- Conventional commit messages: https://www.conventionalcommits.org/en/v1.0.0/



Code structure

- Look for the base class

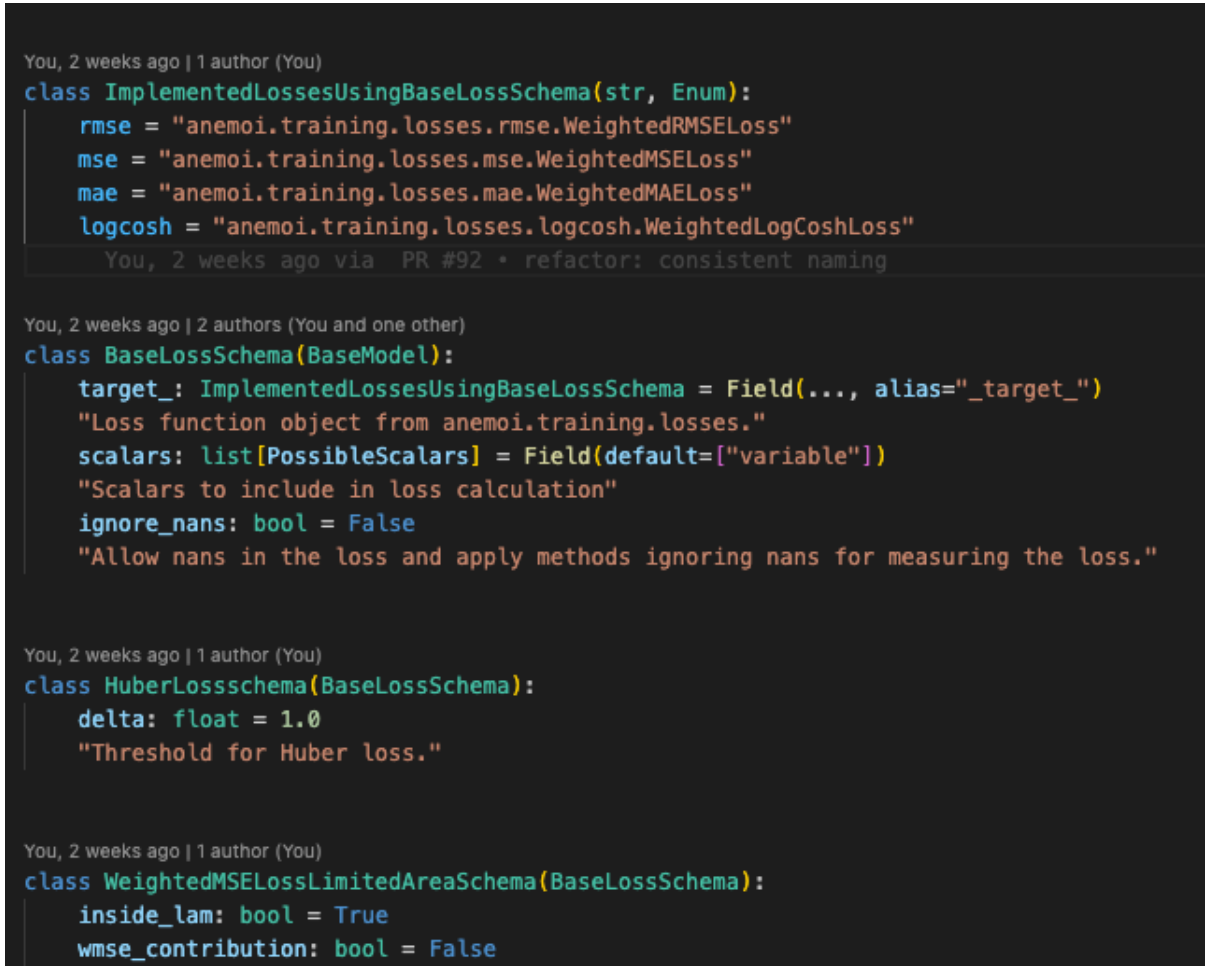



```python
class BaseWeightedLoss(nn.Module, ABC):
    """Node-weighted general loss."""

    scalar: ScaleTensor

    def __init__(
        self,
        node_weights: torch.Tensor,
        ignore_nans: bool = False,
    ) -> None:
        """Node- and feature_weighted Loss.

        Exposes:
        - self.avg_function: torch.nanmean or torch.mean
        - self.sum_function: torch.nansum or torch.sum
        depending on the value of `ignore_nans`

        Registers:      Harrison Cook, 3 months ago • Feature/improve loss functions (#70) …
        - self.node_weights: torch.Tensor of shape (N, )
        - self.scalar: ScaleTensor modified with `add_scalar` and `update_scalar`

        Parameters
        ----------
        node_weights : torch.Tensor of shape (N, )
            Weight of each node in the loss function
        ignore_nans : bool, optional
            Allow nans in the loss and apply methods ignoring nans for measuring the loss, by default False

        """
        super().__init__()

        self.scalar = ScaleTensor()

        self.avg_function = torch.nanmean if ignore_nans else torch.mean
        self.sum_function = torch.nansum if ignore_nans else torch.sum

        self.register_buffer("node_weights", node_weights, persistent=True)
```

Subclass from BaseWeightedLoss

```python
class WeightedRMLELoss(BaseWeightedLoss):
    """Node-weighted RMLE loss."""

    name = "wrmle"

    def __init__(
        self,
        node_weights: torch.Tensor,
        ignore_nans: bool = False,
        **kwargs,
    ) -> None:

        super().__init__(
            node_weights=node_weights,
            ignore_nans=ignore_nans,
            **kwargs,
        )

    def forward(
        self,
        pred: torch.Tensor,
        target: torch.Tensor,
        squash: bool = True,
        scalar_indices: tuple[int, ...] | None = None,
        without_scalars: list[str] | list[int] | None = None,
    ) -> torch.Tensor:
        """Calculates the lat-weighted RMSE loss."""

        out = torch.square(torch.log(pred + 1) - torch.log(target + 1))
        breakpoint()
        out = self.scale(out, scalar_indices, without_scalars=without_scalars)
        return self.scale_by_node_weights(out, squash)
```

Configuration and Schemas



```
You, 2 weeks ago | 1 author (You)
class ImplementedLossesUsingBaseLossSchema(str, Enum):
    rmse = "anemoi.training.losses.rmse.WeightedRMSELoss"
    mse = "anemoi.training.losses.mse.WeightedMSELoss"
    mae = "anemoi.training.losses.mae.WeightedMAELoss"
    logcosh = "anemoi.training.losses.logcosh.WeightedLogCoshLoss"
      You, 2 weeks ago via  PR #92 • refactor: consistent naming

You, 2 weeks ago | 2 authors (You and one other)
class BaseLossSchema(BaseModel):
    target_: ImplementedLossesUsingBaseLossSchema = Field(..., alias="_target_")
    "Loss function object from anemoi.training.losses."
    scalars: list[PossibleScalars] = Field(default=["variable"])
    "Scalars to include in loss calculation"
    ignore_nans: bool = False
    "Allow nans in the loss and apply methods ignoring nans for measuring the loss."


You, 2 weeks ago | 1 author (You)
class HuberLossschema(BaseLossSchema):
    delta: float = 1.0
    "Threshold for Huber loss."


You, 2 weeks ago | 1 author (You)
class WeightedMSELossLimitedAreaSchema(BaseLossSchema):
    inside_lam: bool = True
    wmse_contribution: bool = False
```

Testing

Write tests for functionality

```python
def test_forward_pass(basic_inputs) -> None:
    pred, target, node_weights = basic_inputs
    loss_function = WeightedMSLELoss(node_weights)
    loss = loss_function(pred, target)

    assert isinstance(loss, torch.Tensor)
    assert_close(loss, torch.tensor(0.0, device=loss.device))


def test_weighted_forward_pass(device) -> None:
    pred = torch.tensor([[[[1.0, 2.0], [3.0, 4.0]]]], device=device, requires_grad=True)
    target = torch.tensor([[[[2.0, 2.0], [3.0, 4.0]]]], device=device)
    node_weights = torch.tensor([1.0, 2.0], device=device)
    loss_function = WeightedMSLELoss(node_weights)
    loss = loss_function(pred, target)

    assert loss.item() > 0
    loss.backward()
    assert pred.grad is not None
```

Create Inputs for Tests

```python
@pytest.fixture
def basic_inputs(device) -> tuple:
    pred = torch.tensor([[[[1.0, 2.0], [3.0, 4.0]]]], requires_grad=True, device=de
    target = torch.tensor([[[[1.0, 2.0], [3.0, 4.0]]]], device=device)
    node_weights = torch.ones([2], device=device)
    return pred, target, node_weights
```

Parametrise tests where possible

```python
@pytest.mark.parametrize("ignore_nans", [True, False])
def test_ignore_nans(device, ignore_nans):
    loss_fn = WeightedMRLELoss(ignore_nans=ignore_nans)
    ...
    ...
```

Update documentation

losses.rst

```
**********************
 Default Loss Functions
**********************

By default anemoi-training trains the model using a latitude-weighted
mean-squared-error, which is defined in the ``WeightedMSELoss`` 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:

-  ``WeightedMSELoss``: Latitude-weighted mean-squared-error.
-  ``WeightedMAELoss``: Latitude-weighted mean-absolute-error.
-  ``WeightedHuberLoss``: Latitude-weighted Huber loss.
-  ``WeightedLogCoshLoss``: Latitude-weighted log-cosh loss.
-  ``WeightedRMSELoss``: Latitude-weighted root-mean-squared-error.
-  ``WeightedRMLELoss``: Latitude-weighted root-mean-log-error.      You, 1
-  ``CombinedLoss``: Combined component weighted loss.
```

```
$ cd docs
$ make html
```

View html in _build folder

Default Loss Functions

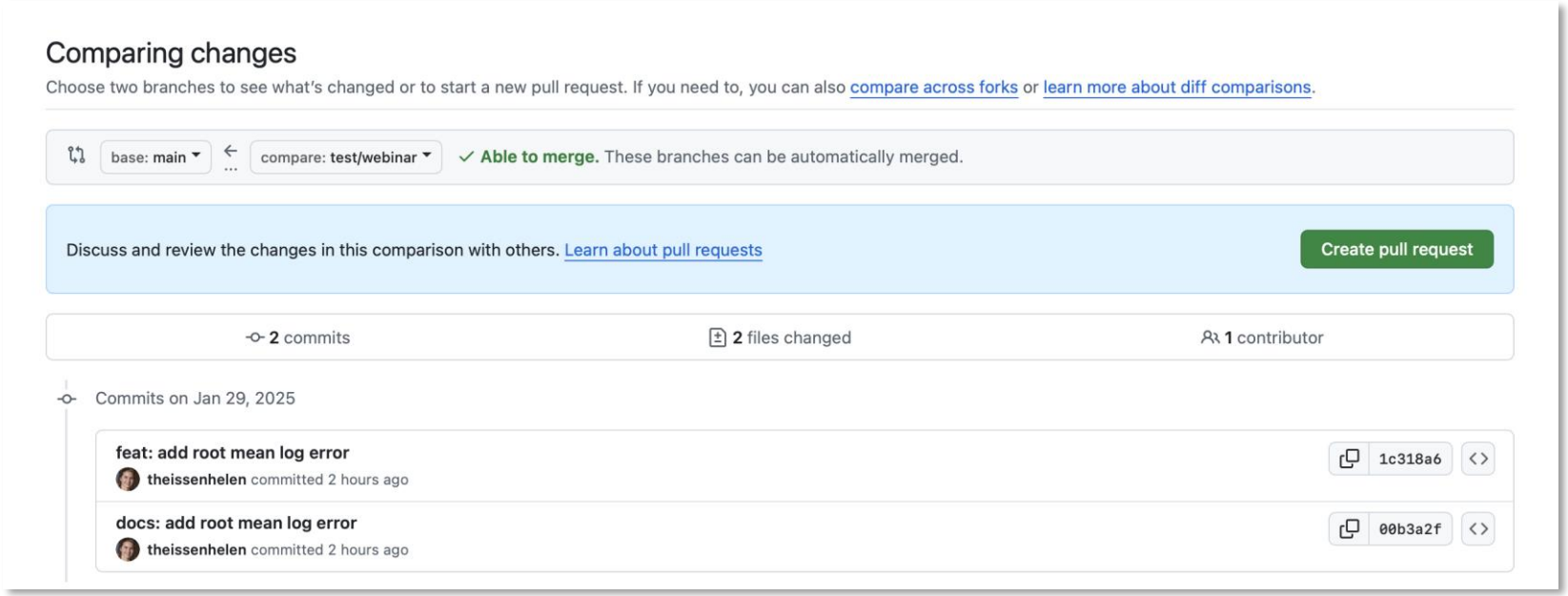
By default anemoi-training trains the model using a latitude-weighted mean-squared-error, which is defined in the `WeightedMSELoss` class in `anemoi/training/losses/mse.py`. The loss function can be configured in the config file at `config.training.training_loss`, and `config.training.validation_metrics`.

The following loss functions are available by default:

- `WeightedMSELoss`: Latitude-weighted mean-squared-error.
- `WeightedMAELoss`: Latitude-weighted mean-absolute-error.
- `WeightedHuberLoss`: Latitude-weighted Huber loss.
- `WeightedLogCoshLoss`: Latitude-weighted log-cosh loss.
- `WeightedRMSELoss`: Latitude-weighted root-mean-squared-error.
- `WeightedRMLELoss`: Latitude-weighted root-mean-log-error.
- `CombinedLoss`: Combined component weighted loss.

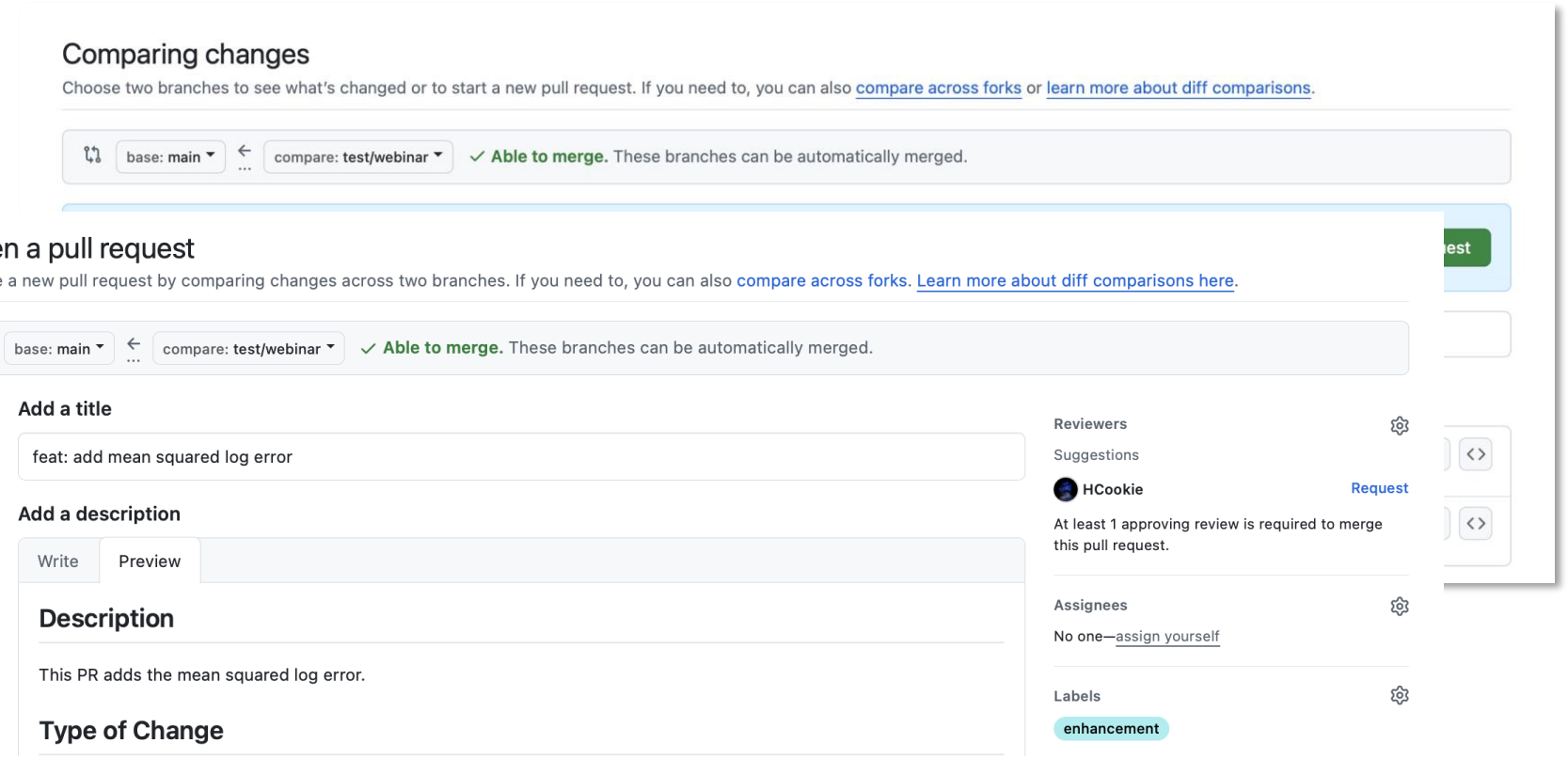
PRs

- PR against the default branch, currently main



PRs

- PR against the default branch, currently main



Template for Pull Requests

Description

Type of Change

- [ ] Bug fix (non-breaking change which fixes an issue)
- [ ] New feature (non-breaking change which adds functionality)
- [ ] Breaking change (fix or feature that would cause existing functionality to not work as expected)
- [ ] Documentation update

Issue Number

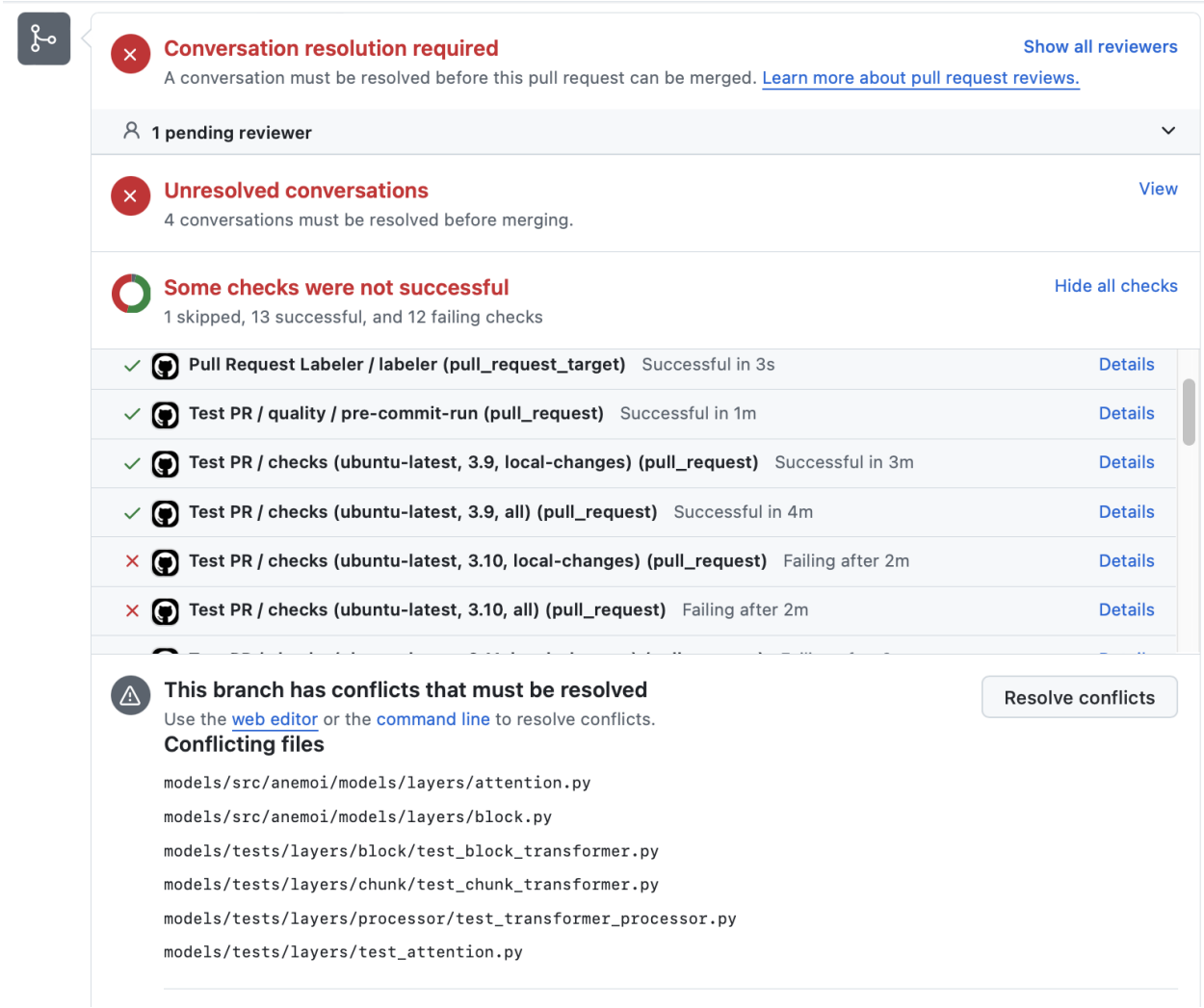
Code Compatibility

- [ ] I have performed a self-review of my code

Code Performance and Testing

- [ ] I have added tests that prove my fix is effective or that my feature works

CI/CD – Continuous Integration / Continuous development



Summary and Best practices

- Start with an issue to facilitate discussions
- Prepare a PR that links to the issue
- Always run pre-commit
- Implement subclasses from appropriate base classes for new features
- Anemoi follows PEP 8 guidelines for python code
- Inline comments for more complex logic
- Write tests for new features

Useful commands

- Config generation

```
$ anemoi-training config generate
```

- Config validation

```
$ anemoi-training config validate -config-name=debug
```

- Running training

```
$ anemoi-training train -config-name=debug
```

- Running pytests

```
$ pytest training
```

Questions?

anemoi-graphs.readthedocs.io