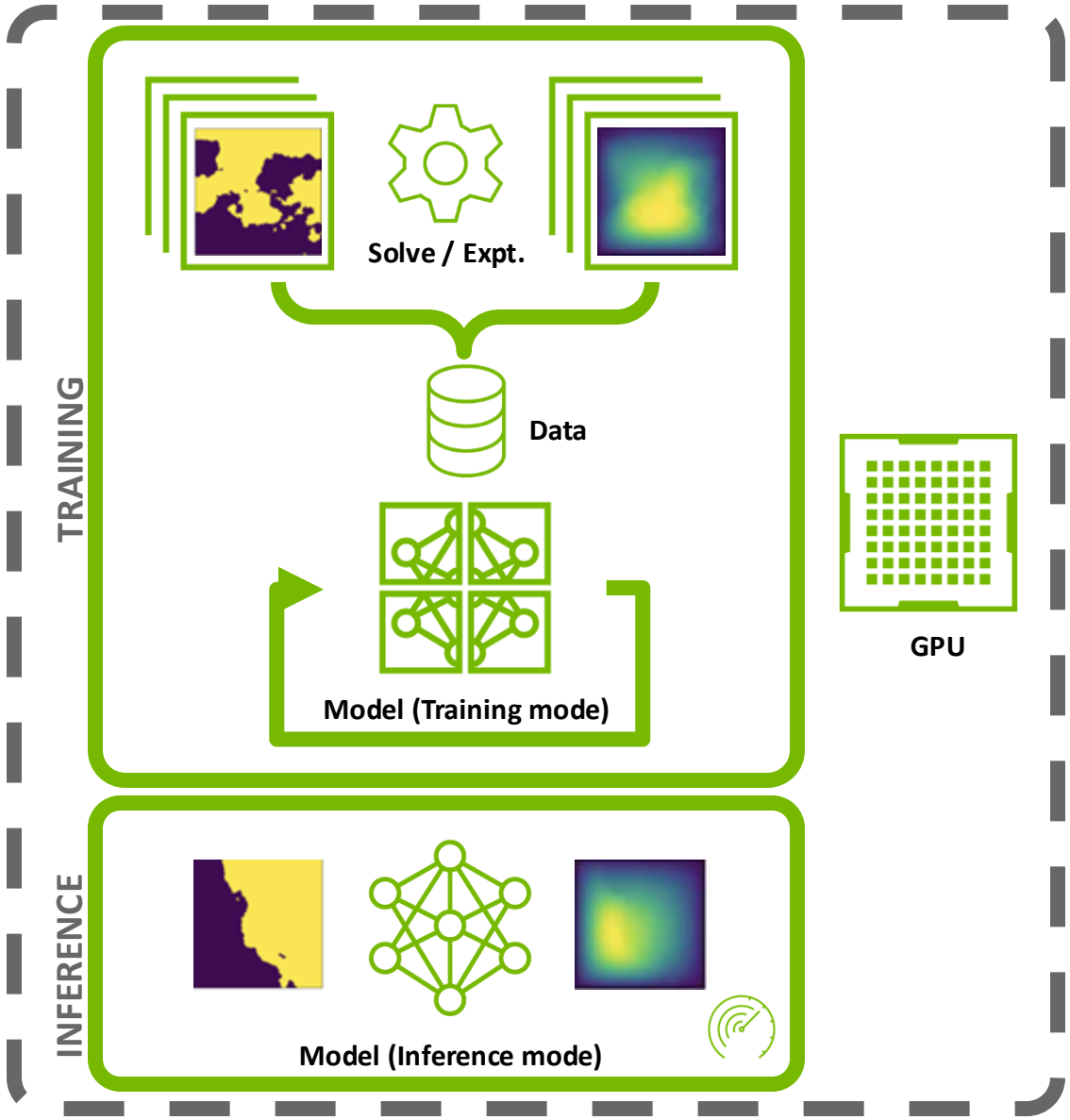
NVIDIA PhysicsNeMo: An open-source deep learning framework for physical systems

Niki Loppi
nloppi@nvidia.com

Developing Physics-ML models using data in PhysicsNeMo

- Train a data-driven model:
  - Given input-output pairs, find the map (operator)
  - Instantaneous field –> another field
  - Initial condition –> steady solution (steady state surrogate models)
  - Step -> next step (autoregressive 1 step solvers)

- Define the training loop using interoperable PhysicsNeMo-PyTorch utilities
  - Optimized data loaders and data pipes for domain specific use cases
  - Optimized Model Zoo and layers targeted for Physics-ML applications
  - Utilities for Distributed training and scaling
    - Easily apply performance optimizations like CUDA Graphs, AMP
  - Utilities for easier logging
  - Checkpointing utilities to save and load checkpoints
    - Load model checkpoints without model arguments
  - Utilities for deployment and inference



Developing Physics-ML models using data in PhysicsNeMo

Typical training workflow

Python Imports

```python
import hydra
from omegaconf import DictConfig
import torch
import numpy as np
import matplotlib.pyplot as plt
from hydra.utils import to_absolute_path
import torch.nn.functional as F
from torch.utils.data import DataLoader
from itertools import chain

from physicsnemo.models.fno import FNO
from physicsnemo.launch.logging import LaunchLogger
from physicsnemo.launch.utils.checkpoint import save_checkpoint

from utils import HDF5MapStyleDataset

@hydra.main(version_base="1.3", config_path="conf",
config_name="config_pure_data.yaml")
def main(cfg: DictConfig):
```

Data loading

```python
    LaunchLogger.initialize()

    dataset = HDF5MapStyleDataset(
        to_absolute_path("./datasets/Darcy_241/train.hdf5"), device="cuda"
    )
    validation_dataset = HDF5MapStyleDataset(
        to_absolute_path("./datasets/Darcy_241/validation.hdf5"), device="cuda"
    )

    dataloader = DataLoader(dataset, batch_size=2, shuffle=True)

    validation_dataloader = DataLoader(validation_dataset, batch_size=1, shuffle=False)
```

Model

```python
    model_branch = FNO(
        in_channels=2,
        out_channels=1,
     ).to("cuda")
```

Optimizer

```python
    optimizer = torch.optim.Adam(
        chain(model_branch.parameters()),
        betas=(0.9, 0.999),
        lr=cfg.start_lr,
        weight_decay=0.0,
    )

    scheduler = torch.optim.lr_scheduler.ExponentialLR(optimizer, gamma=cfg.gamma)
```

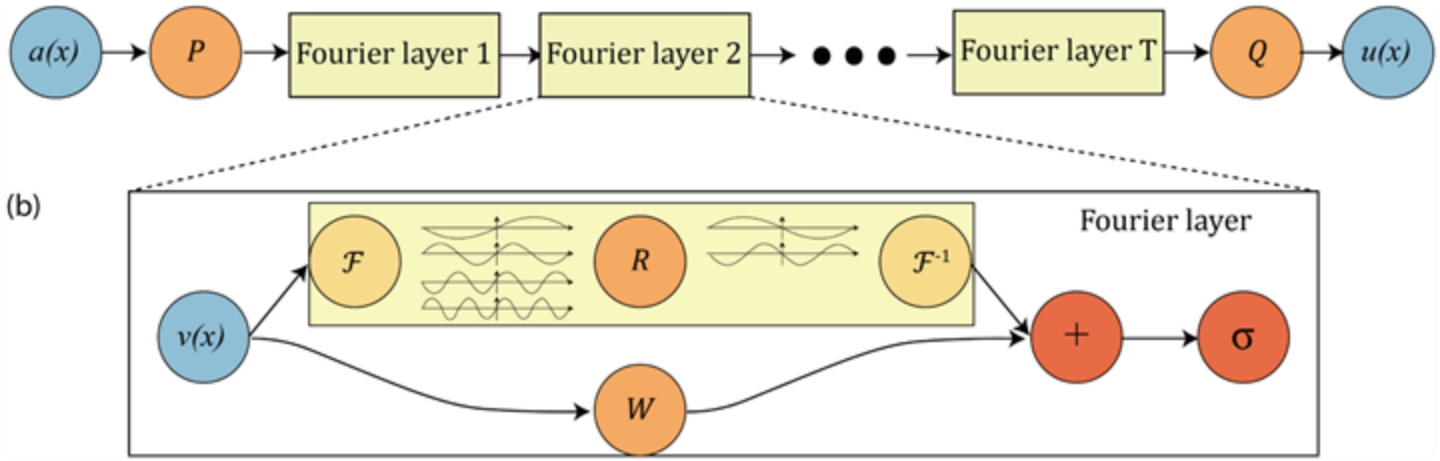
Training loop with loss computation

```python
    for epoch in range(cfg.max_epochs):
        # wrap epoch in launch logger for console logs
        with LaunchLogger(
            "train",
            epoch=epoch,
            num_mini_batch=len(dataloader),
            epoch_alert_freq=10,
        ) as log:
            for data in dataloader:
                optimizer.zero_grad()
                truevar = data[1]

                # compute forward pass
                outvar = model_branch(data[0][:, 0].unsqueeze(dim=1))

                # Compute data loss
                loss_data = F.mse_loss(outvar, truevar)

                # Compute total loss
                loss = loss_data

                # Backward pass and optimizer and learning rate update
                loss.backward()
                optimizer.step()
                scheduler.step()

            log.log_epoch({"Learning Rate": optimizer.param_groups[0]["lr"]})

        # test model on validation dataset
        with LaunchLogger("valid", epoch=epoch) as log:
            error = validation_step(model_branch, validation_dataloader, epoch)
            log.log_epoch({"Validation error": error})
```

Saving model

```python
        # save the checkpoint
        save_checkpoint(
            "./checkpoints",
            models=[model_branch],
            optimizer=optimizer,
            scheduler=scheduler,
            epoch=epoch,
        )


if __name__ == "__main__":
    main()
```

Fourier Neural Operators



- Mapping between function spaces (operator)
  - Continuous, mesh-independent, resolution-invariant
- Global convolutions

$$r(x) = \{u * v\}(x) = \mathcal{F}^{-1}\{U \cdot V\}.$$

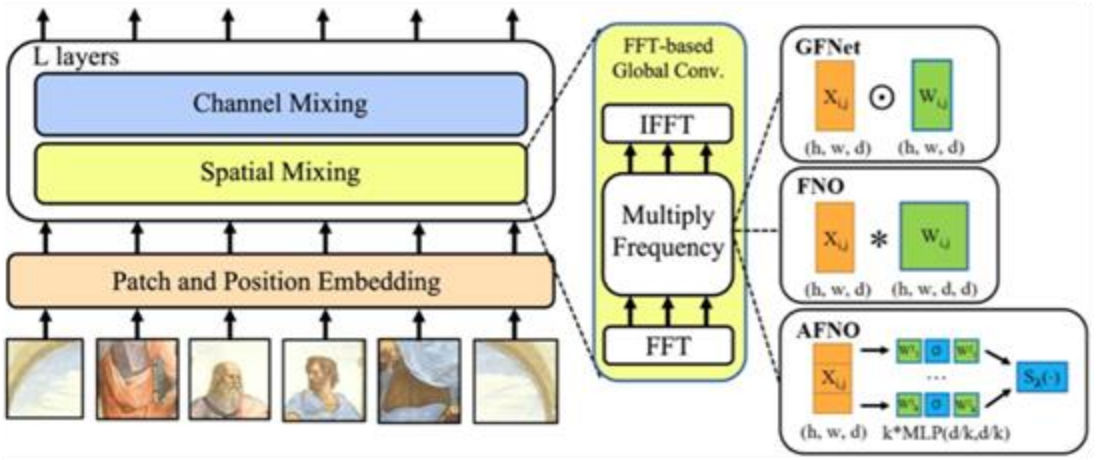
- Numerically efficient
  - Clip “excessive” high freq modes

Li, Kovachki, et al. (2021), Fourier neural operator for parametric partial differential equations, ICLR 2021

Guibas, Mardani, et al. (2022), Efficient token mixing for transformers via

Adaptive Fourier Neural Operators

Inspiration from Vision Transformers



Guibas, Mardani, et al. (2022), Efficient token mixing for transformers via Adaptive FNO, ICLR 2022

- Significantly more parameter efficient
- Patch original inputs and embed
  - E.g. Strided conv2d with embed dim channels
- Spectral conv similar to FNO
  - spatial mixing, global (using emb patches)
- MLP channel wise (channel mix) and output

Developing Physics-ML models using data in PhysicsNeMo

Using FNOs to create a surrogate model

- Develop a surrogate model that learns a mapping between a permeability field and pressure field, $\mathbf{K} \rightarrow \mathbf{U}$, for a distribution of permeability fields $\mathbf{K} \sim p(\mathbf{K})$.
- The Darcy PDE (Diffusion equation) is a second order, elliptical PDE with the following form:

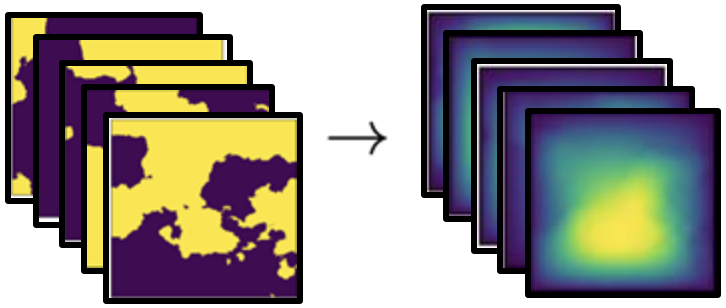
$$-\nabla \cdot \big(k(\mathbf{x}) \nabla u(\mathbf{x}) \big) = f(\mathbf{x}), \qquad \mathbf{x} \in D,$$

  - $u(\mathbf{x})$: Flow pressure
  - $k(\mathbf{x})$: Permeability field
  - $f(\cdot)$: Forcing function
  - $D = \{x, y \in (0,1)\}$ with the boundary condition $u(\mathbf{x}) = 0, \mathbf{x} \in \partial D$.
- Both the permeability and flow fields are discretized into a 2D matrix $\mathbf{K}, \mathbf{u} \in \mathbb{R}^{N \times N}$.

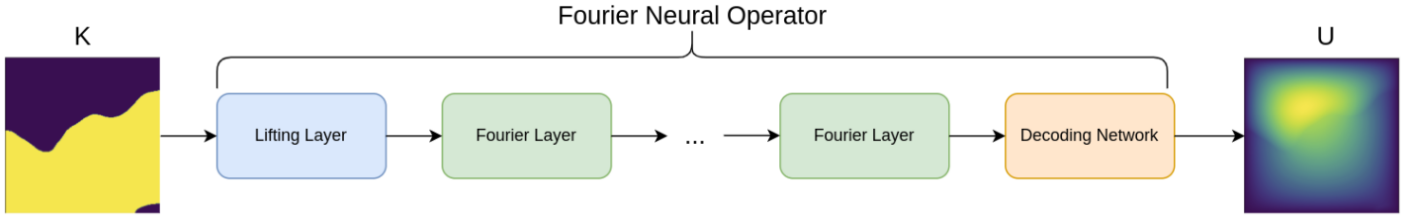
$$\vec{v} = -\frac{K}{\mu\epsilon}\vec{\nabla}p$$

$$\vec{\nabla} \cdot v = \vec{\nabla} \cdot \left(\frac{K}{\mu\epsilon}\vec{\nabla}p\right)$$

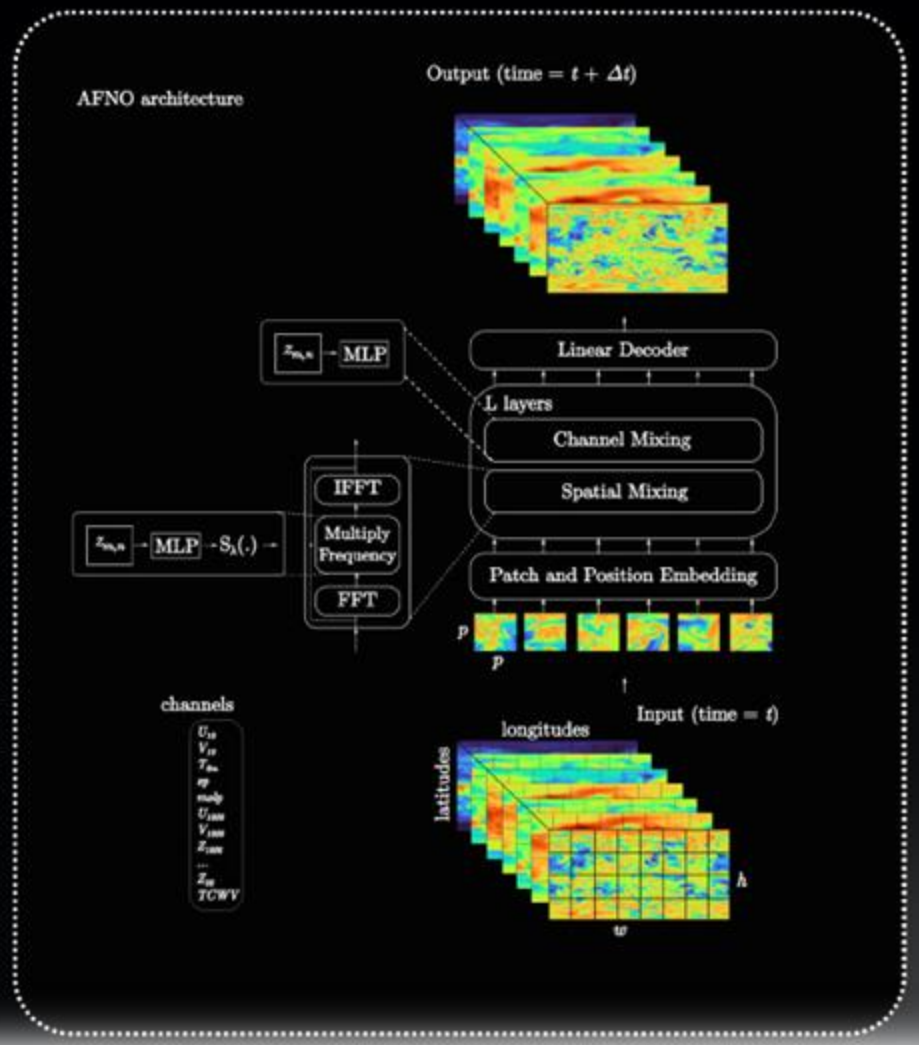
$$\vec{\nabla} \cdot v = 0$$



Input: coefficients (k) Output: solutions (u)



FOURCASTNET (FOURIER FORECASTING NETWORK)



Purely data-driven machine learning surrogate weather model

Trained on ERA5 reanalysis data at the native resolution of 0.25 degrees.

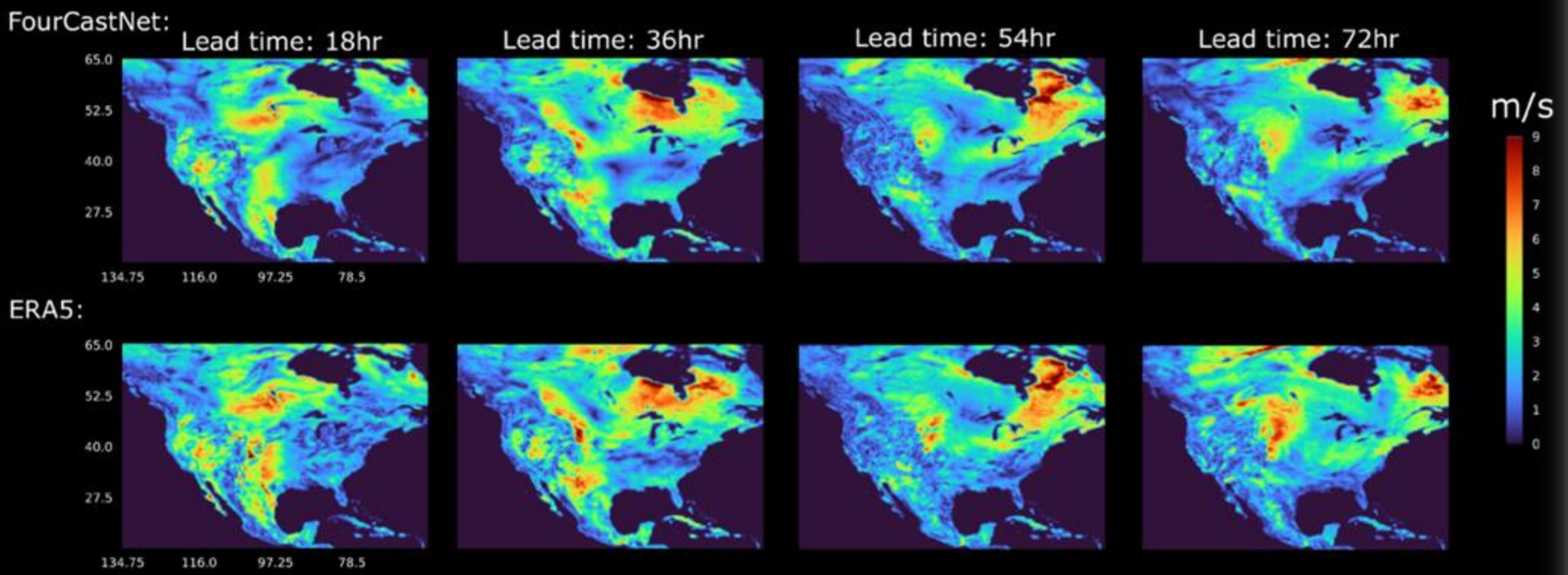
State-of-the-art for Deep Learning based weather surrogate models.

Highest resolution data driven model ever trained.

Guibas et al. (2022), Adaptive Fourier Neural Operators: Efficient Token Mixers for Transformers, ICLR 2022.

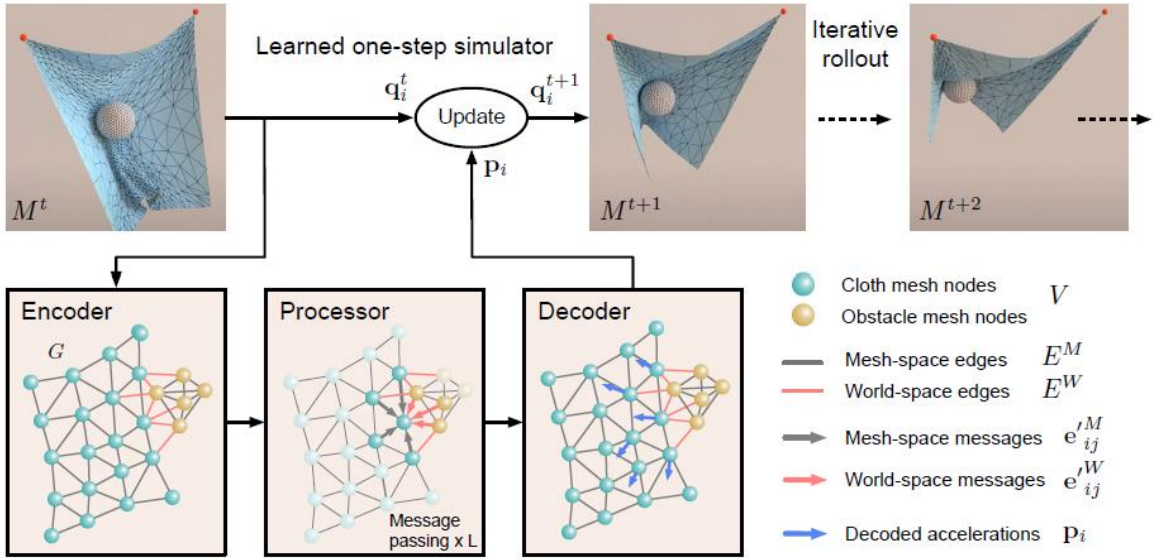
Pathak et al. (2022), FourCastNet: A Global Data-driven High-resolution Weather Model using Adaptive Fourier Neural Operators, arXiv:2202.11214

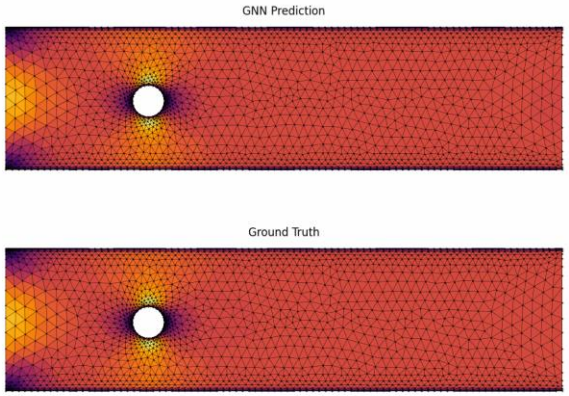
FOURCASTNET PREDICTS NEAR-SURFACE WIND FIELDS OVER LAND ACCURATELY: IMPORTANT IMPLICATIONS FOR WIND ENERGY PLANNING



Graph Neural Networks

Leveraging the mesh





- Mesh-based simulations are central to modeling in many disciplines across science and engineering.
- GNNs are a natural choice for data-driven approaches – its mesh-based and can cope with geometry irregularities and multi-scale physics.
- Forward model – Predict variables of the mesh at time t+1 given current mesh at t and history of previous meshes.

CorrDiff

12.5x super-resolution + radar channel synthesis

FEATURES:
ERA5:
36 x 36 (20-ch)

TARGETS:
Radar-assimilating WRF
448x448 (4-ch)