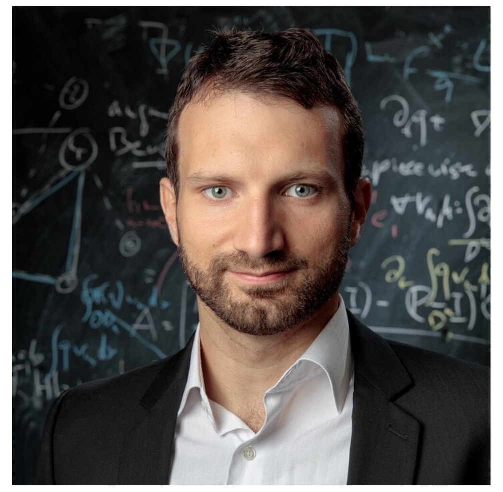
Spherical Fourier Neural Operators

Boris Bonev | Large-scale deep learning for Earth Systems | Bonn, 22. August 2023

FourCastNet team

Members of the NVIDIA Earth-2 initiative and academic collaborators


Boris Bonev
NVIDIA

Thorsten Kurth
NVIDIA


Christian Hundt
NVIDIA


Jaideep Pathak
NVIDIA


Maximilian Baust
NVIDIA


Karthik Kashinath
NVIDIA


Anima Anandkumar
NVIDIA / Caltech


Kamyar Azizzadenashli
NVIDIA


Morteza Mardani
NVIDIA


Mike Pritchard
NVIDIA


Noah Brenowitz
NVIDIA


Yair Cohen
NVIDIA


Jean Kossaifi
NVIDIA


David Pruitt
NVIDIA


Nikola Kovachki
NVIDIA


Shashank S.
LBL


Sanjeev R.
U Michigan

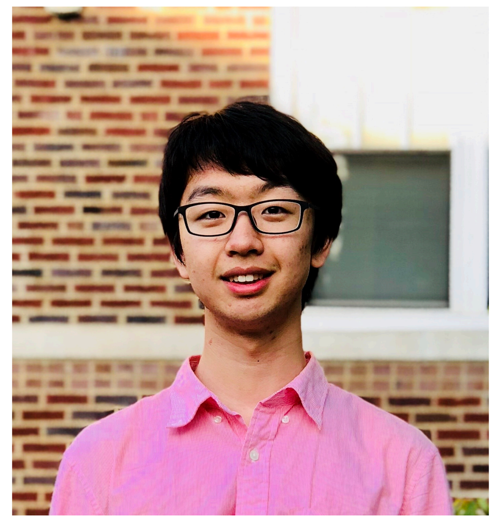

Peter H.
LBL


Pedram H.
Rice U.


Justin Luitjens
NVIDIA


Ashesh C.
Rice U.

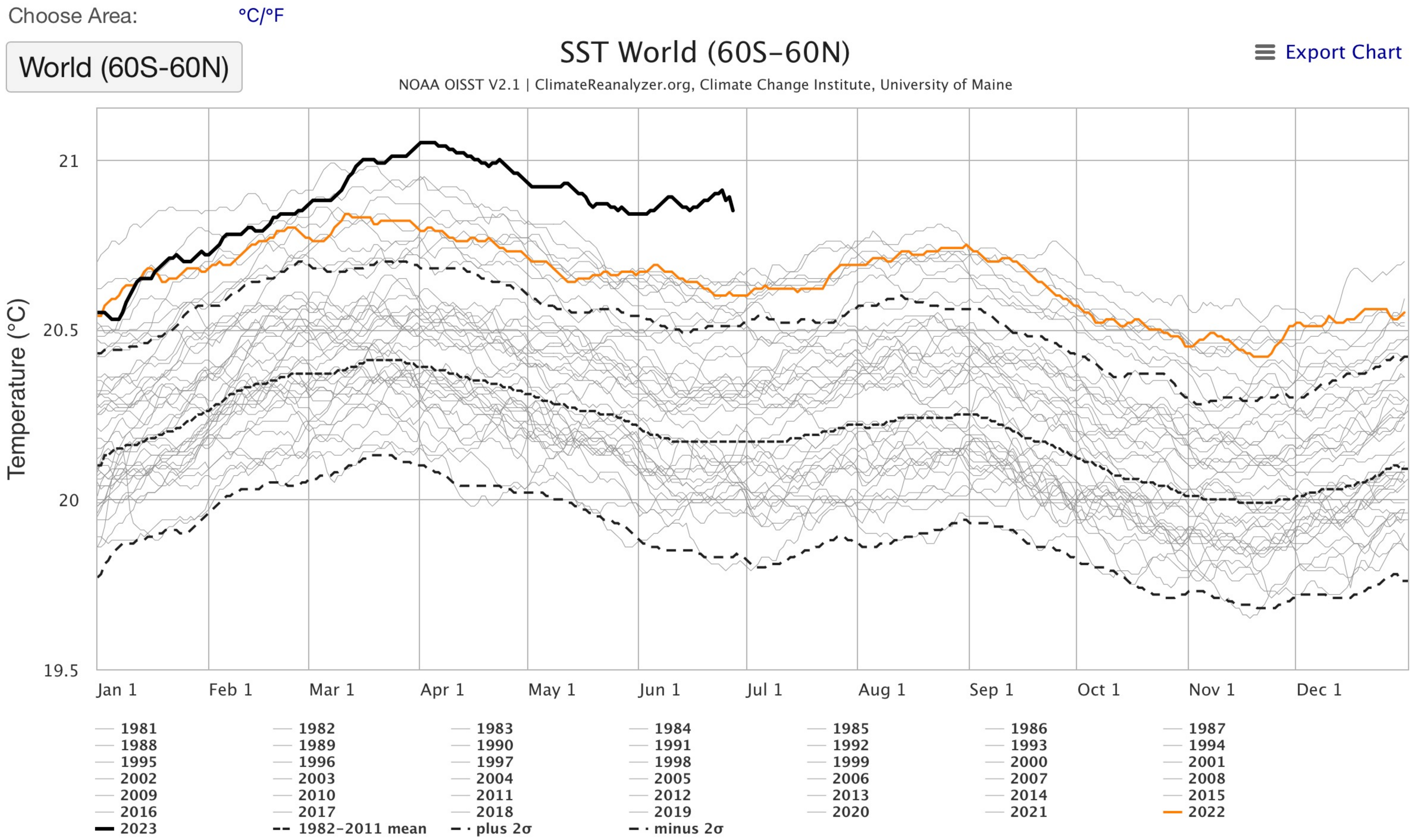

David H.
NVIDIA


Zongyi L.
Caltech

Motivation

Climate change

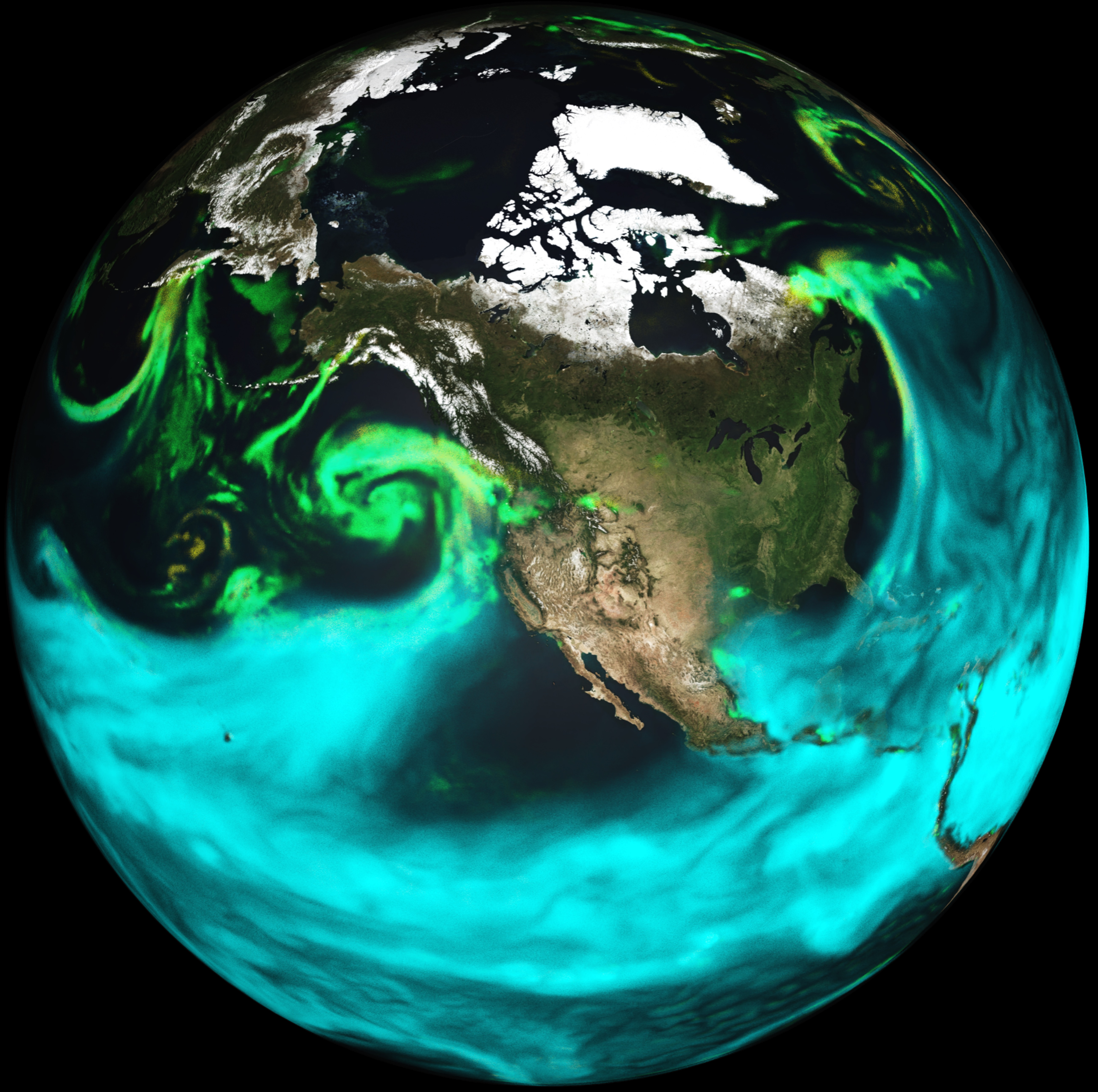
increased likelihood of extreme weather events



Source: NOAA – ClimateReanalyzer.org, 29.6.2023

Realistic climate simulation is a computational grand challenge

Figure adapted from: Schneider, T., Teixeira, J., Bretherton, C. *et al.* "*Climate goals and computing the future of clouds*". *Nature Climate Change* **7,** 3–5 (2017)



Why ML-based Weather Predicition?

- large speedups:
  - Not constrained to timestep restrictions such as CFL conditions
  - better suited to uniform memory-access patterns on GPUs
- E2E calibration is straightforward
- Quality data available (ECMWF's ERA5 Reanalysis dataset)
- FourCastnet, PanguWeather, GraphCast and others have deomonstrated this
- **However:** how can we trust such data-driven methods, if they are not built from first principles?

Spherical Fourier Neural Operators

Setting

ML-based weather prediction

Earth's atmosphere is modelled as a dynamical system::

$$u_{n+1} = F[u_n, t_n]$$

- u_n vector-valued state of the atmosphere at time t_n
- F maps this state to the next state u_{n+1} at time t_{n+1}.
- F is typically obtained by integrating a PDE in time.
- This often depends on the discretization of u_n.
- In other cases, such as weather, F is unknown due to the complexity of the system.

ML approach: obtain an approximation $\tilde{F}$ from data, i.e:

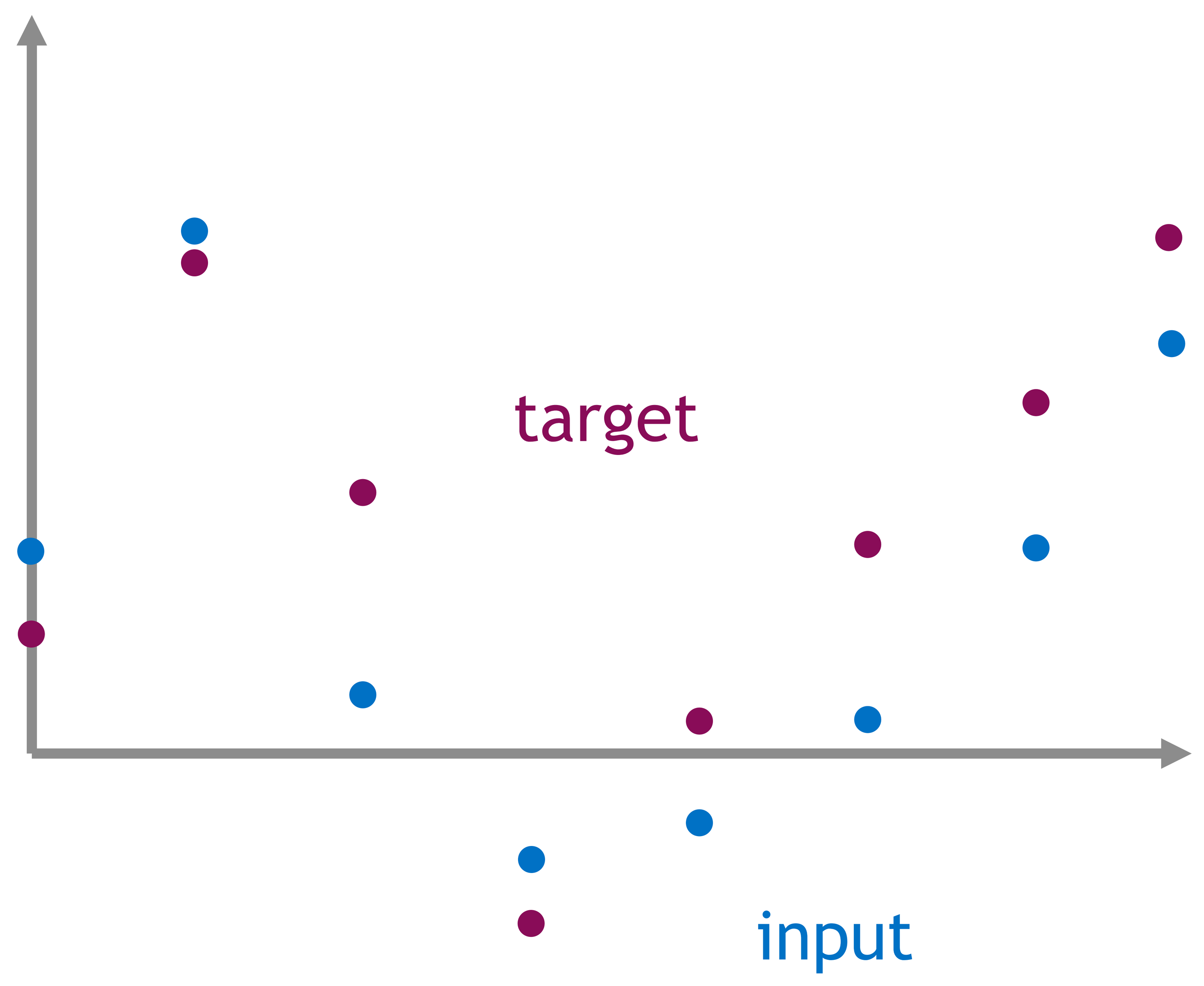
$$u_{n+1} = \tilde{F}[u_n,\ t_n;\ \theta]$$

Fourier Neural Operators

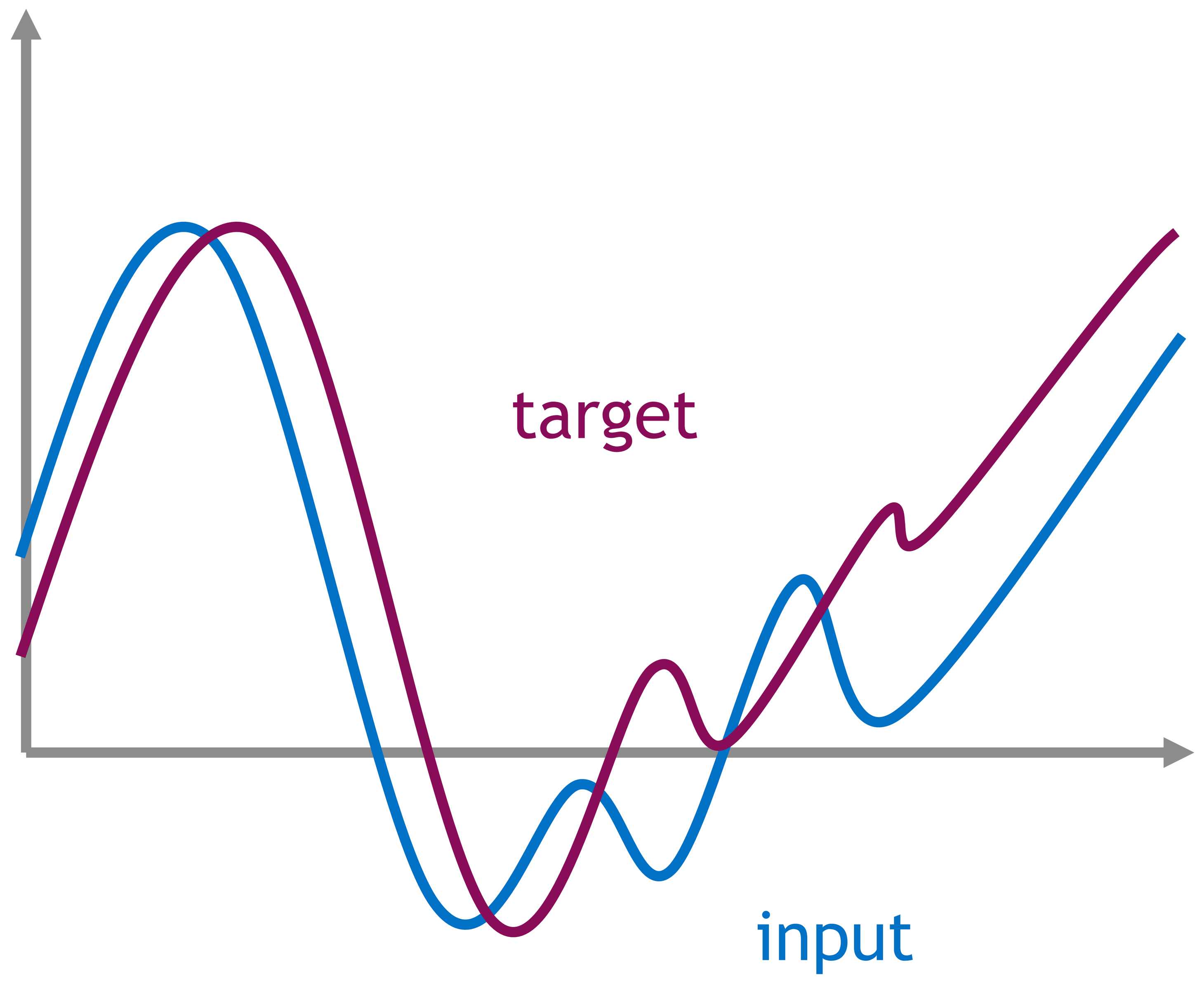
motivation

$$u_{n+1} = \tilde{F}[u_n, t_n;\ \theta]$$

F is fundamentally a functional map from one functions space to another. Neural operators treat u as a function rather than a data-vector which depends on the chosen discretization.



what a conventional network sees



What a Neural Operator sees

Fourier Neural Operators

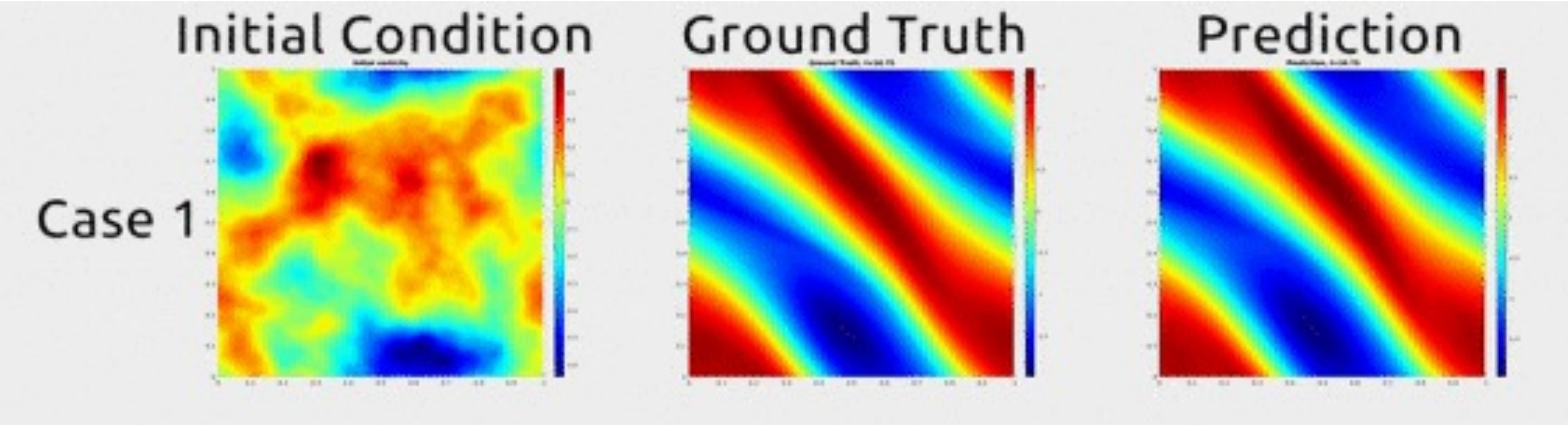
how to define Neural maps between function spaces

The key piece in FNOs are the Fourier layers which are (global) convolutions with a learned filter κ :

$$K[u](x) = \int_{\Omega} \kappa(y)\, u(x-y)\, dy\,.$$

In practice, we can leverage the convolution theorem to compute this efficiently via FFTs:

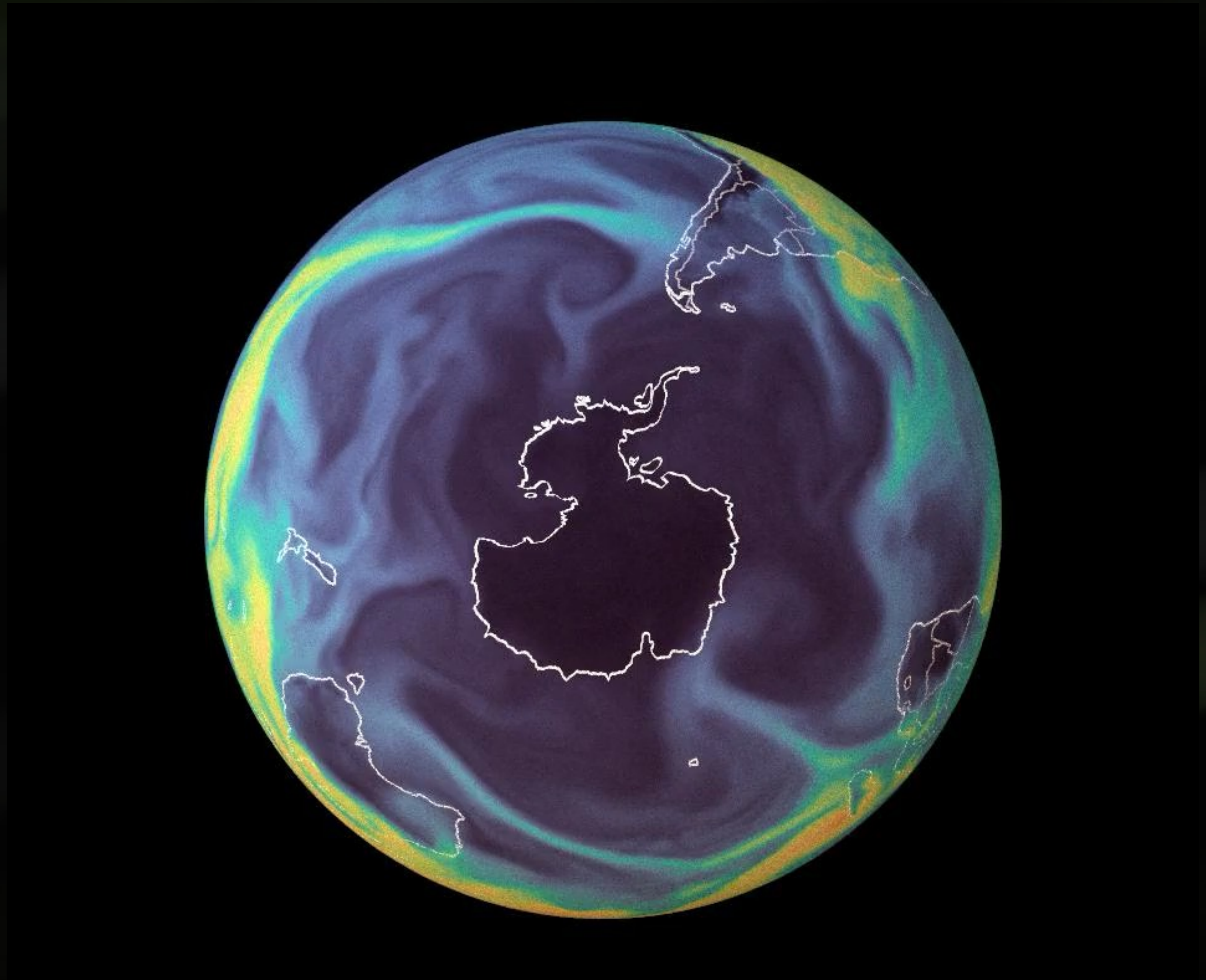
$$K[u] = \mathcal{F}^{-1}\left[\mathcal{F}[\kappa] \cdot \mathcal{F}[u]\right]$$



Li et al. 2020: Fourier Neural Operator for Parametric Partial Differential Equation

The problem: Polar Instabilities

AFNO is treating spatial domain incorrectly



Correct topology is S^2 and not $S^1 \times S^1$
(autoregressive feedback loop amplifies small errors over rollout steps)

Why symmetries matter

Equivariance of the dynamical system

Earth's atmosphere is modelled as a dynamical system where F is learned from data:

$$u_{n+1} = F[u_n]$$

Most physical systems do not change as we change our frame of reference. On the sphere, we can express this as follows:

$$RF[u] = F[Ru],$$

where R is a rotation operator acting on functions. Expressing the system in a different frame of reference leaves the equation unchanged:

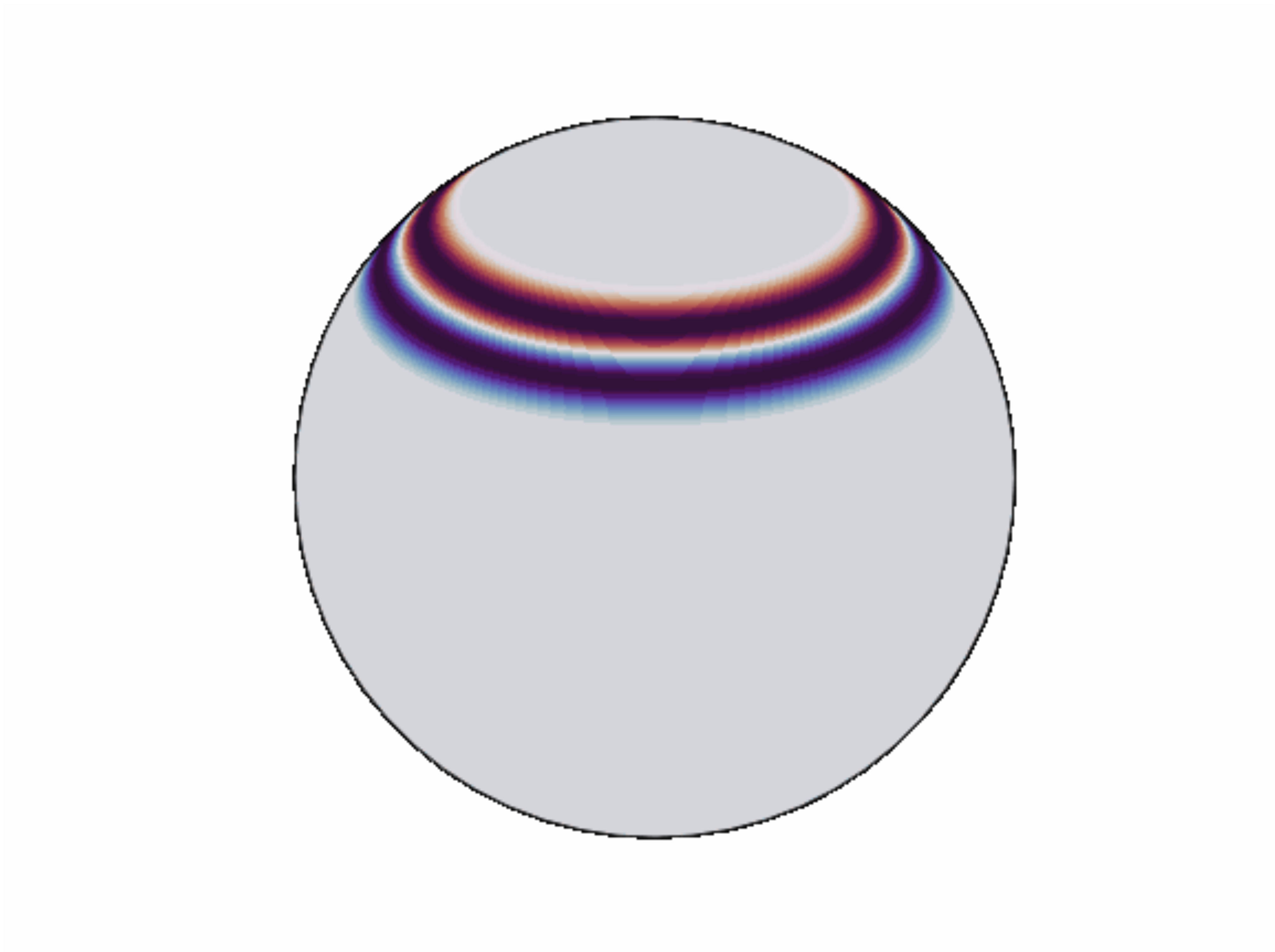
$$Ru_{n+1} = u'_{n+1} = F[u'_n] = RF[u_n].$$

This acts as a valuable inductive bias for our ML model.

In physics, the evidence is clear: **symmetries are the key!**

Why symmetries matter

Symmetry in the spherical Shallow Water Equations



$$\begin{cases} \frac{\partial \boldsymbol{q}}{\partial t} + \boldsymbol{\nabla} \cdot \boldsymbol{F}(\boldsymbol{q}) = \boldsymbol{S}(\boldsymbol{x}, \boldsymbol{q}) & \text{in } \Omega \times (0, \infty) \\ \boldsymbol{q} = \boldsymbol{q}_0 & \text{on } \Omega \times \{t = 0\}, \end{cases}$$

State vector:

$$\boldsymbol{q} = \begin{bmatrix} \varphi & \varphi u & \varphi v & \varphi w \end{bmatrix}^{\mathsf{T}},$$

Gradient of the Flux is rotationally equivariant (believe me!):

$$\begin{aligned} \boldsymbol{F}(\boldsymbol{q}) &= \boldsymbol{f}_x(\boldsymbol{q})\ \hat{\boldsymbol{i}} + \boldsymbol{f}_y(\boldsymbol{q})\ \hat{\boldsymbol{j}} + \boldsymbol{f}_z(\boldsymbol{q})\ \hat{\boldsymbol{k}} \\ &= \begin{bmatrix} \varphi u \\ \varphi u^2 + \frac{1}{2}\varphi^2 \\ \varphi u v \\ \varphi u w \end{bmatrix} \hat{\boldsymbol{i}} + \begin{bmatrix} \varphi v \\ \varphi u v \\ \varphi v^2 + \frac{1}{2}\varphi^2 \\ \varphi v w \end{bmatrix} \hat{\boldsymbol{j}} + \begin{bmatrix} \varphi w \\ \varphi u w \\ \varphi v w \\ \varphi w^2 + \frac{1}{2}\varphi^2 \end{bmatrix} \hat{\boldsymbol{k}}, \end{aligned}$$

Source terms have explicit dependence on x and break the symmetry

$$\tilde{\boldsymbol{S}}(\boldsymbol{x}, \boldsymbol{q}) = \boldsymbol{C}(\boldsymbol{x}, \boldsymbol{u}) - \varphi \boldsymbol{\nabla} \tau(\boldsymbol{x}) + \mu \boldsymbol{x}.$$

$$\boldsymbol{C}(\boldsymbol{x}, \boldsymbol{u}) = -\frac{2\omega z \varphi}{R^2} \boldsymbol{x} \times \boldsymbol{u},$$

However, symmetry is broken only “weakly”, in the sense that the source term can be treated as extra input and rotated with it!

How can we generalize FNOs on the Sphere?

Group actions and convolutions on the Sphere

Which symmetries should we consider?

The sphere is **not** a group, however, rotations in $SO(3)$ “sweep” the sphere. In technical terms, we say that the sphere is an orbit of the subgroup $SO(2)$, i.e. $S^2 = SO(3)/SO(2)$.

Convolutions and their link to group actions

Convolutions are defined as the inner product of two functions, where one function is “shifted” by the **group action** of the underlying Lie group. On the sphere, we can define

$$K[u](x) = \int_{R \in SO(3)} \kappa(Rn)\, u(R^{-1}x)\, dR\,,$$

where κ is the convolution kernel, R a rotation in $SO(3)$ and n the north pole.

If n is replaced with the origin 0, and R with $+y$, we recover the usual convolution theorem!

Convolution theorem on the Sphere

Generalization of FNOs

Based on the convolution theorem, a Fourier transform $\mathcal{F}$ can be defined for the sphere based on the Spherical Harmonic Transform

$$\hat{u}(l,m) = \mathcal{F}[u](l,m) = \int_{S^2} \overline{Y_l^m} \cdot u \, d\Omega,$$

where $\overline{Y_l^m}$ denote the Spherical Harmonics.

The right way of doing spherical geometry is **not** by using the Spherical Harmonic basis but rather the convolution theorem on the sphere:

$$\mathcal{F}\big[K[u]\big](l,m) = \mathcal{F}[\kappa](l,0) \cdot \mathcal{F}[u](l,m)$$

Advantages of this approach:

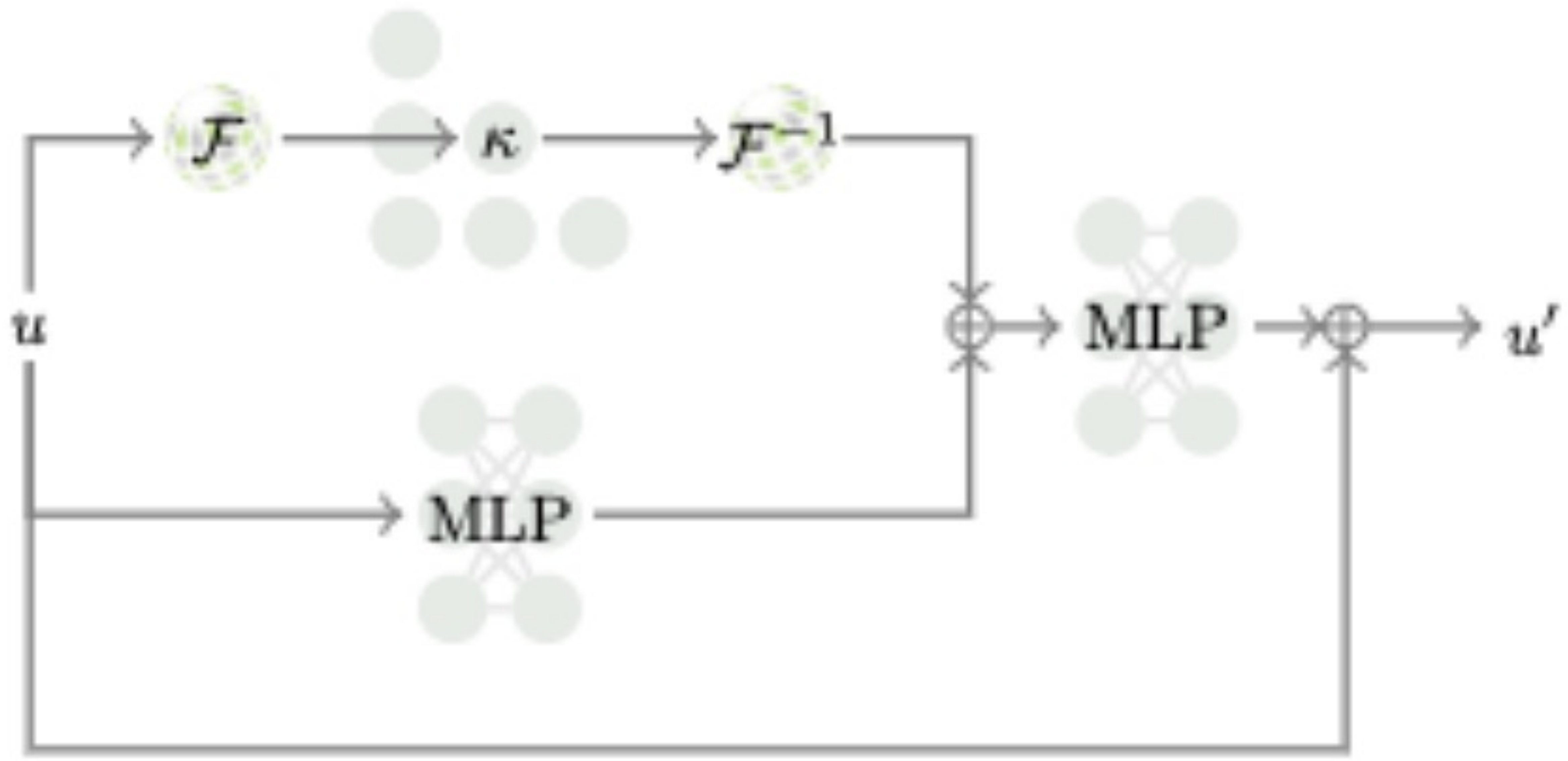
- Grid-invariance
  - Neural operator can be evaluated on **any grid without retraining** as long as SHT can be computed
  - This includes super-resolution and regional forecasting out-of-the-box
- Correct treatment of inherent symmetries
  - Equivariance makes the model similar to the physical processes which satisfy similar symmtry conditions

*Driscoll J., Healy D.; Computing Fourier transforms and convolutions on the 2-sphere. Advances in Applied Mathematics 1994

Formulating Fourier Neural Operators on the Sphere

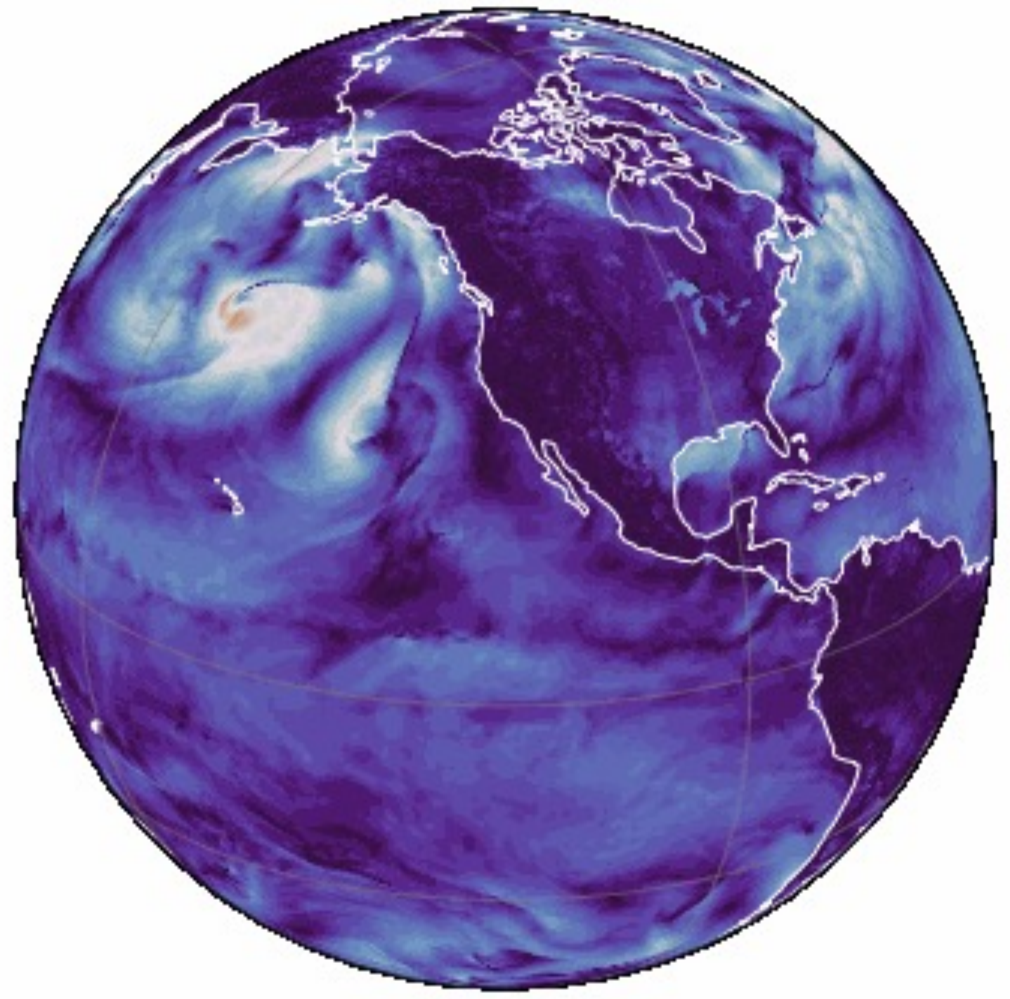
Generalization of FNOs

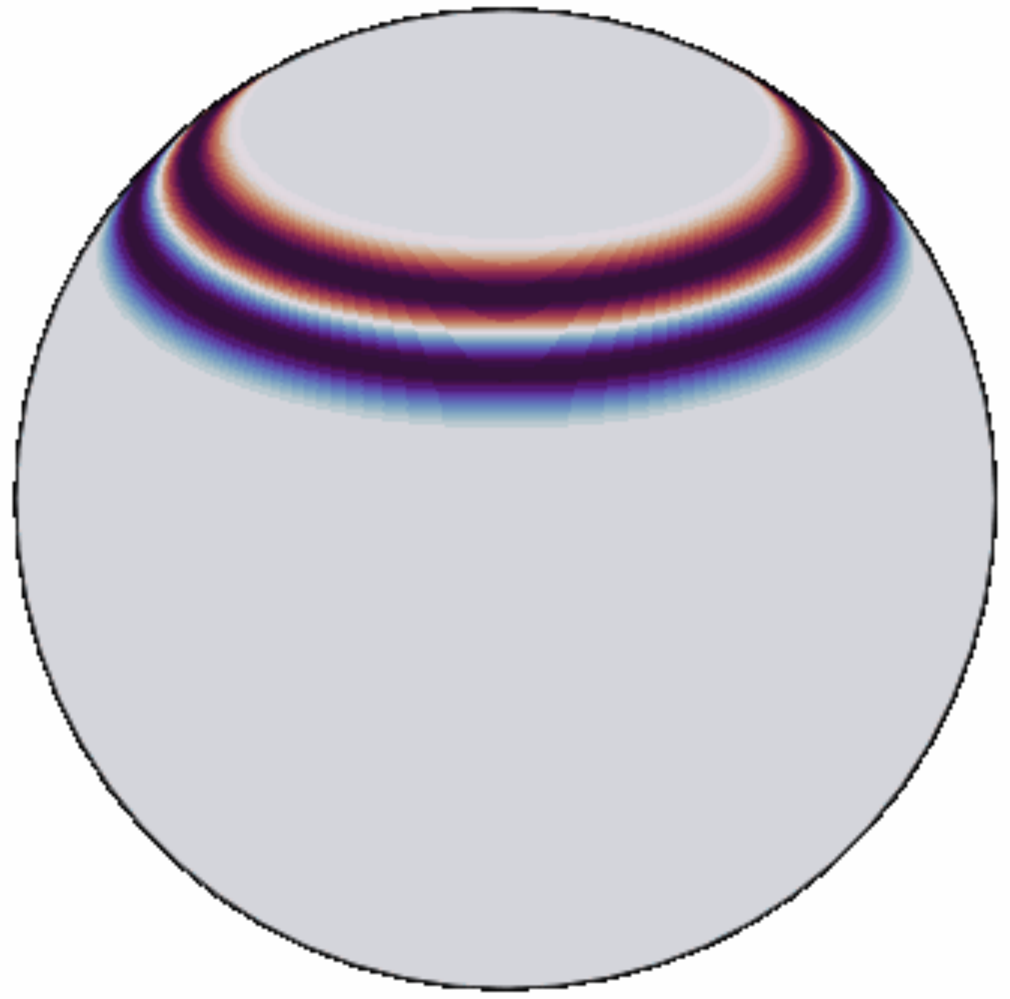
- Encoder and decoder layer are **pointwise MLPs**
- **SFNO blocks** contain spherical convolution and point-wise MLPs
- **Position embedding** models position debendent effects that break the symmetry such as Coriolis forces or orography

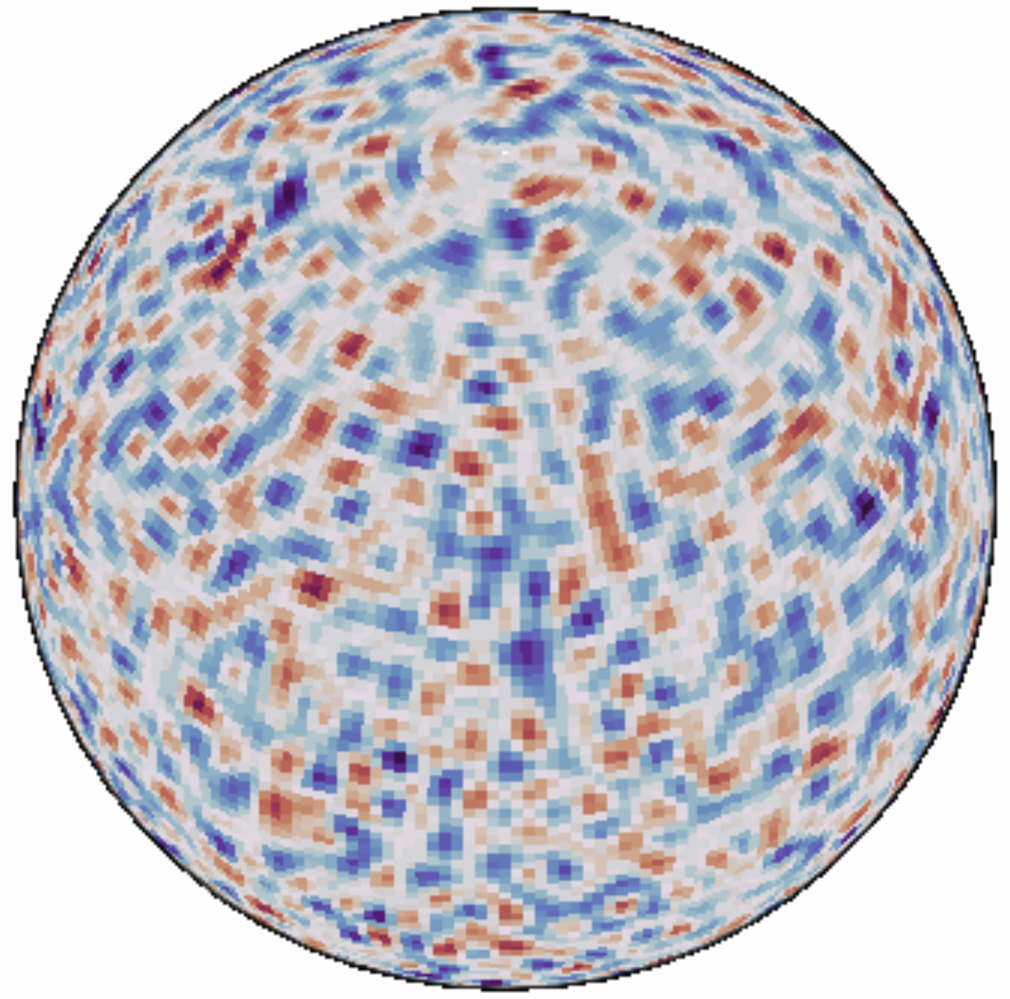


torch-harmonics

A library for differentiable Spherical Harmonics Transforms (SHT)







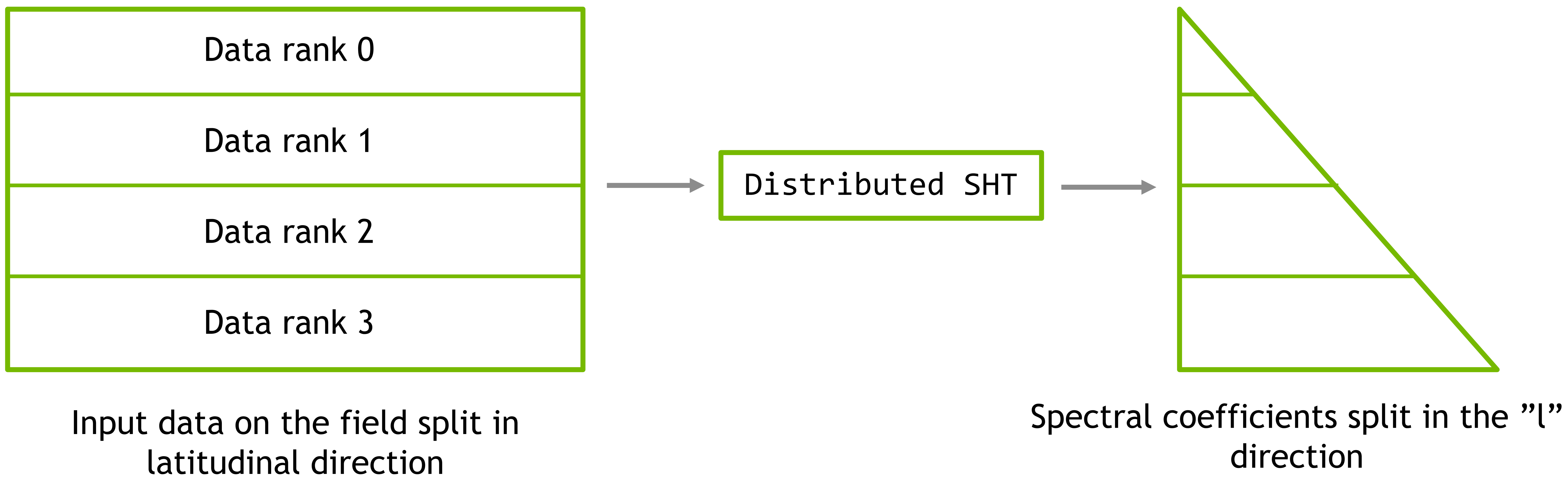
```python
import torch
import torch_harmonics as th

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

nlat = 512
nlon = 2*nlat
batch_size = 32
signal = torch.randn(batch_size, nlat, nlon)

# transform data on an equiangular grid
sht = th.RealSHT(nlat, nlon, grid="equiangular").to(device).float()

coeffs = sht(signal)
```

harmonics

- Open-Source library under MIT license: https://github.com/NVIDIA/torch-harmonics
- Efficient calls for forward and inverse (vector) spherical harmonic transformations
- Autograd support as differential layers in PyTorch
- Distributed computation across many GPUs
- Easy to integrate into existing PyTorch code, contains many examples, among them SFNO

Spatial parallelism in SFNO

Achieving model parallelism via distributed SHT (torch-harmonics)



- Distributed SHT and FFT implementations support other spatial parallelisms such as **simultaenous h/w parallelism**
- Technical implementation is more involved
- Additional measures such as **padding required** to keep tensor equal in size
- **Point-wise operations** are inherently parallel so the rest of the architecture is straight-forward to parallelize
- Dataloaders need to be adapted as well. Spatial parallelism has the added benefit of **reducing I/O**
- SFNO now also supports additional **channel parallelism** (tensor-parallelism)

Results

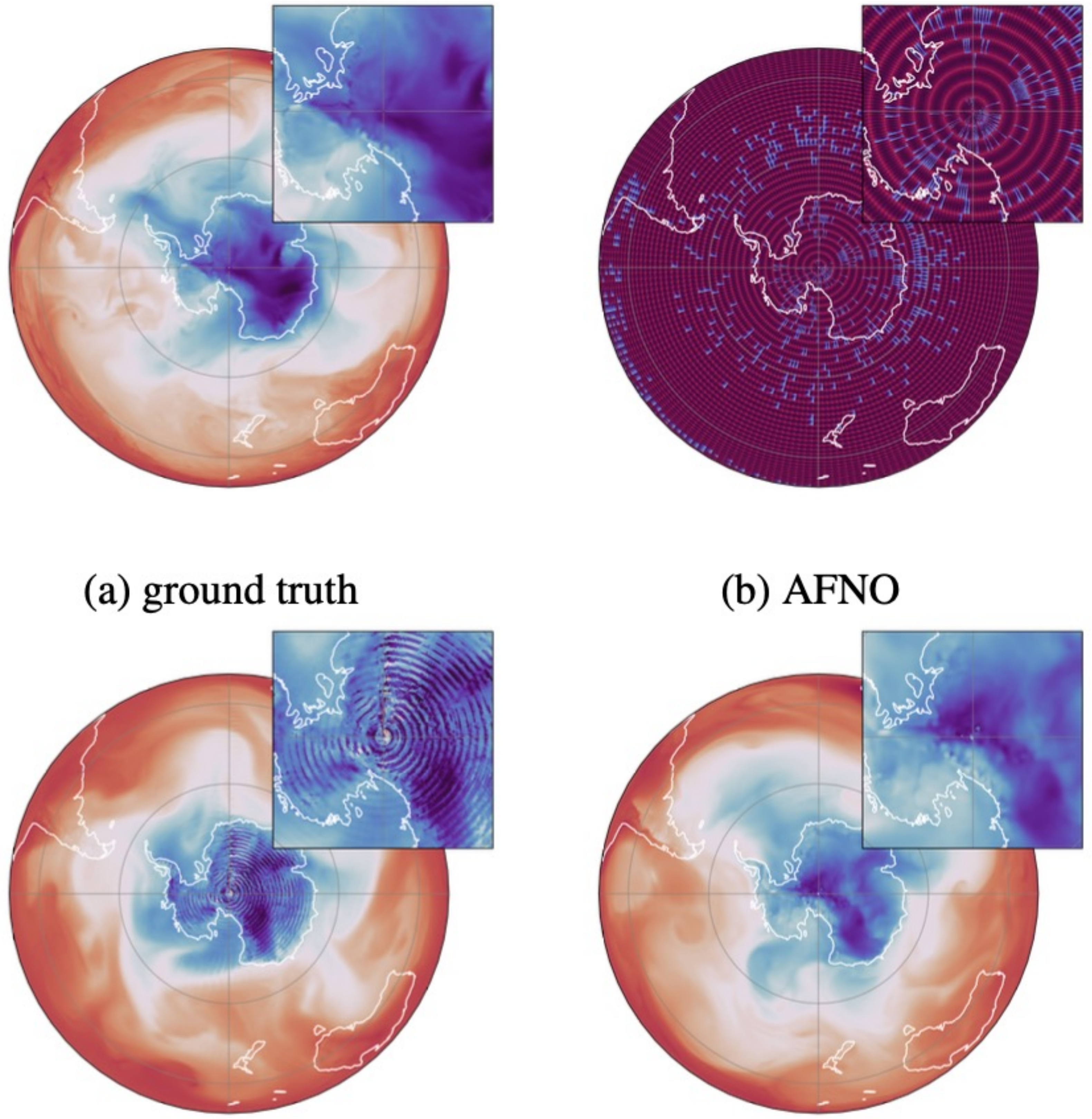
Results on the Spherical Shallow Water Equations

- Our model outperforms the baseline U-Net and the regular FNO on this dataset
- Moreover, it's approximation error (w.r.t. data) is lower than that of the classical method used to generate the data

Model	Parameters			L^2 Loss		Eval time
	Layers	Embed. Dimension	Parameter Count	at 1h (1 step)	at 10h (10 steps)	
U-Net	20	-	$3.104 \cdot 10^7$	$2.961 \cdot 10^{-3}$	$1.462 \cdot 10^{-1}$	0.011s
FNO, linear	4	256	$4.998 \cdot 10^7$	$8.280 \cdot 10^{-4}$	$9.958 \cdot 10^{-3}$	0.156s
FNO, non-linear	4	256	$3.920 \cdot 10^7$	$8.298 \cdot 10^{-4}$	$9.784 \cdot 10^{-3}$	0.212s
SFNO, linear	4	256	$3.518 \cdot 10^7$	$7.741 \cdot 10^{-4}$	$7.239 \cdot 10^{-3}$	0.218s
SFNO, non-linear	4	256	$3.920 \cdot 10^7$	$7.673 \cdot 10^{-4}$	$1.558 \cdot 10^{-2}$	0.321s
Classical Solver	-	-	-	$1.891 \cdot 10^{-2}$	$3.570 \cdot 10^{-2}$	1.299s

Results on the ERA5 dataset

- Results show remarkable stability and an absense of artifacts even during long rollouts (1460 steps).
- Results match the accuracy of the gold-standard in classical weather prediction, IFS at a speedup of 5000x.

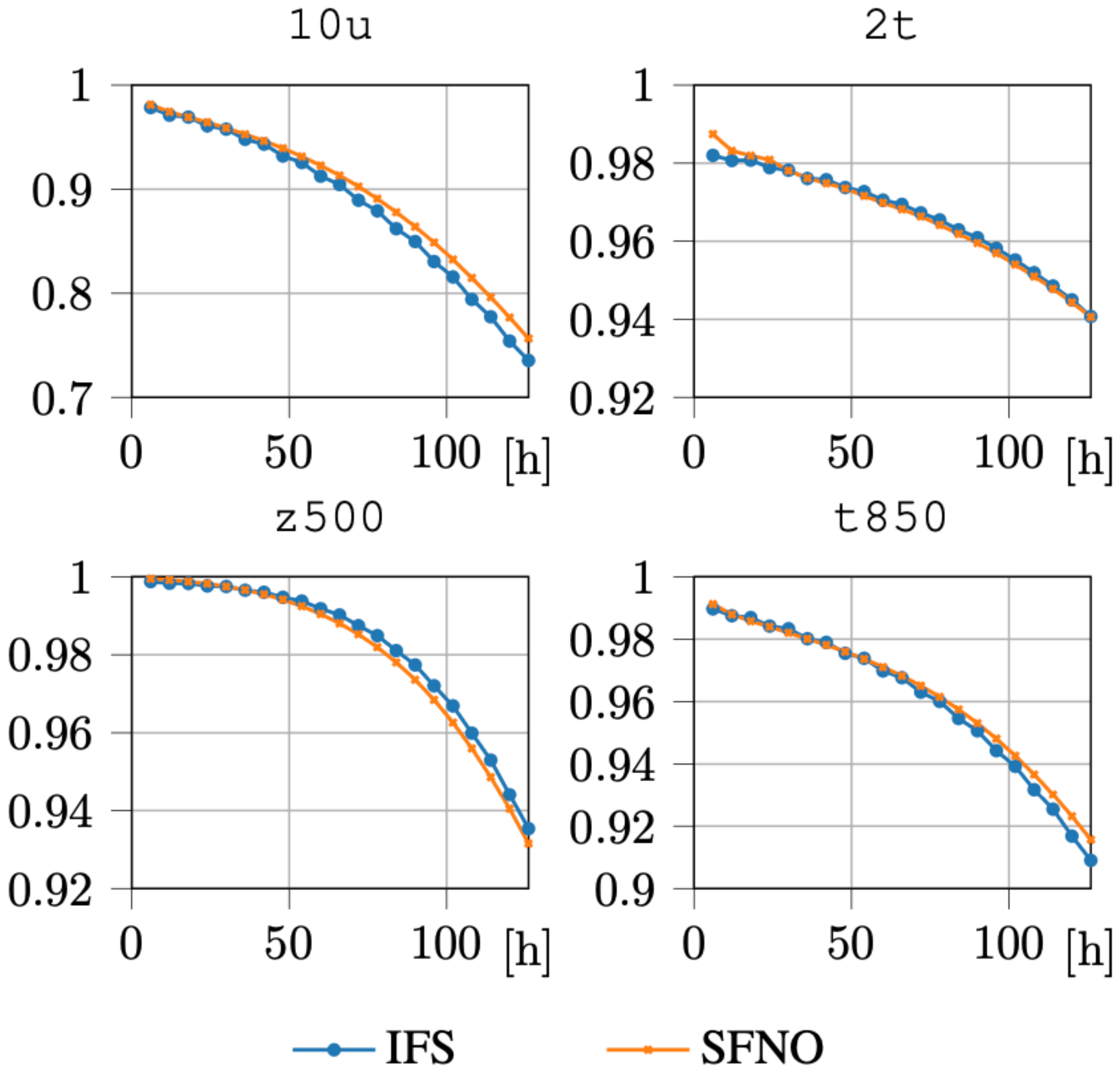


(a) ground truth

(b) AFNO

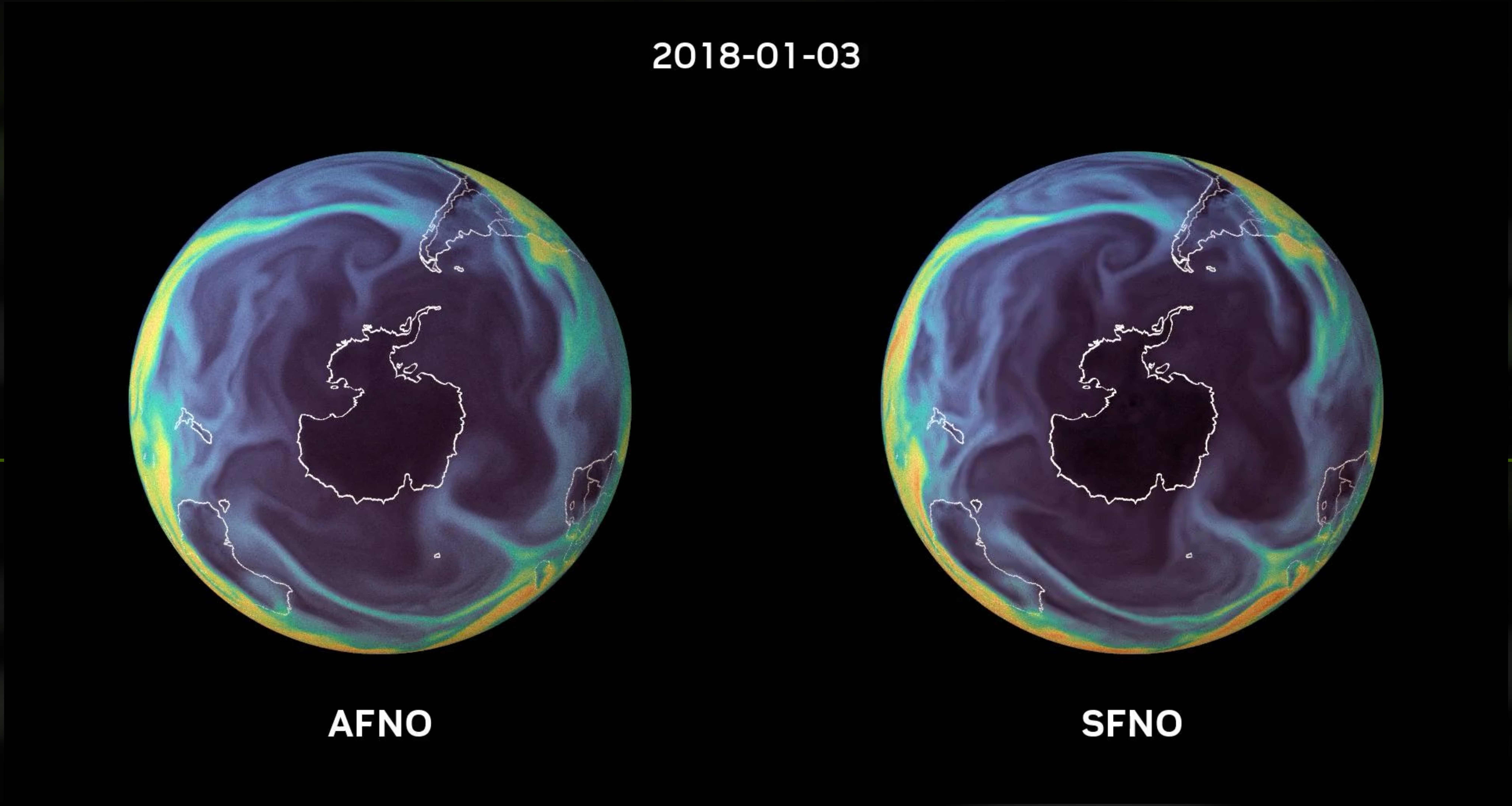
(c) FNO, non-linear

(d) SFNO, linear



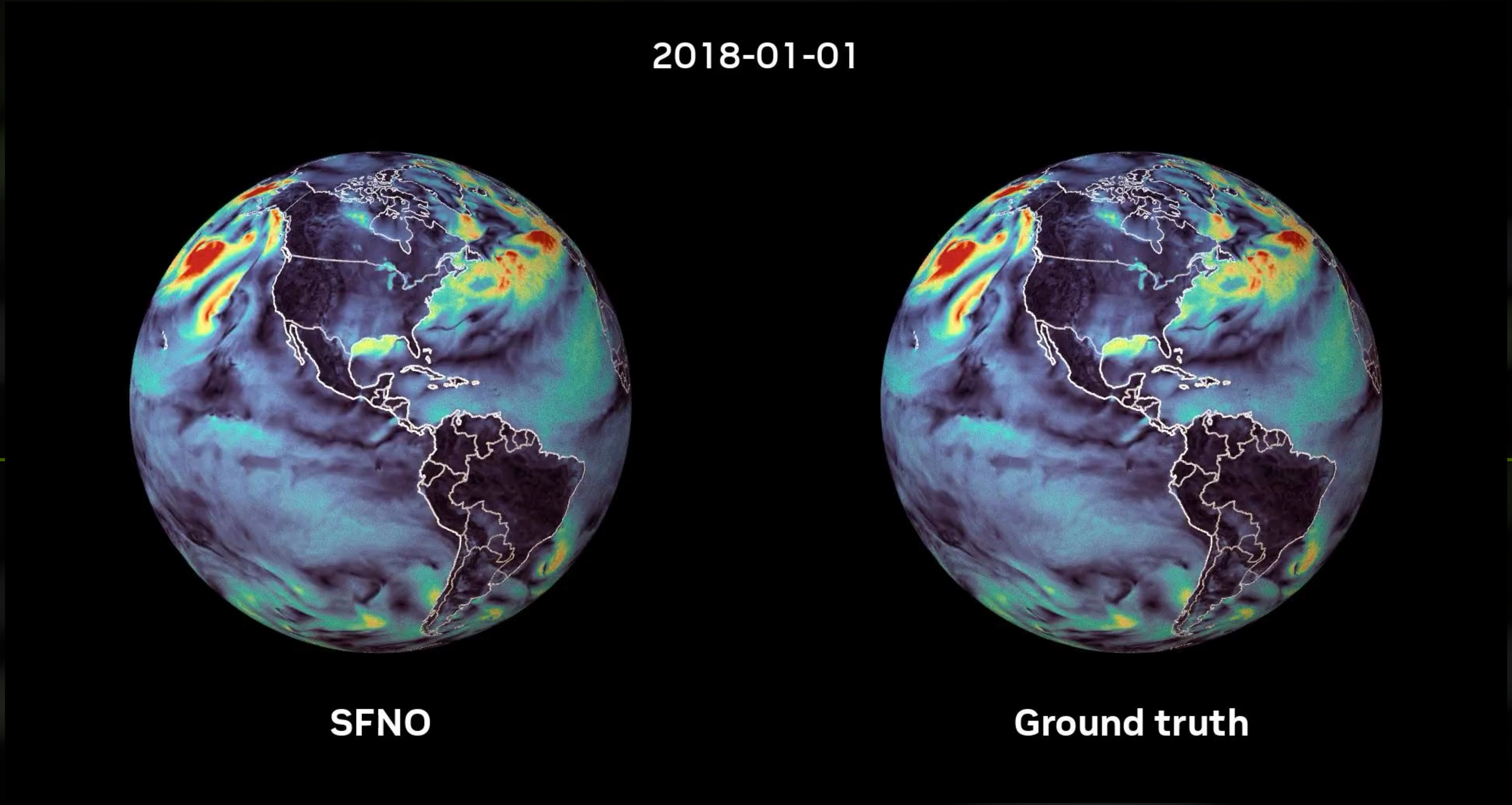
Results for SFNO trained on ERA5

Comparison of rollouts at the polar region

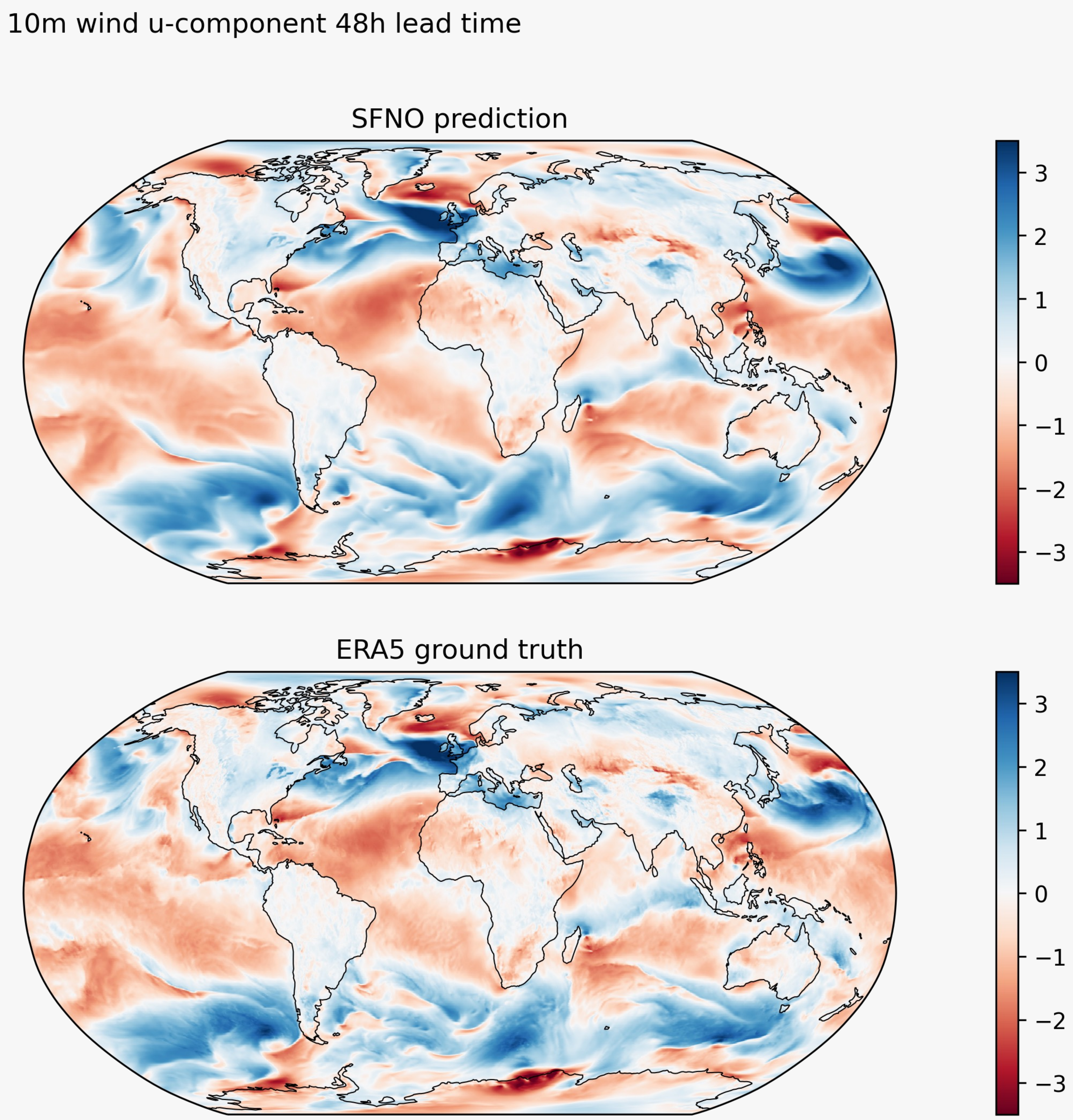


Results for SFNO trained on ERA5

5-month long stable rollout, computed on a single NVIDIA RTX A6000

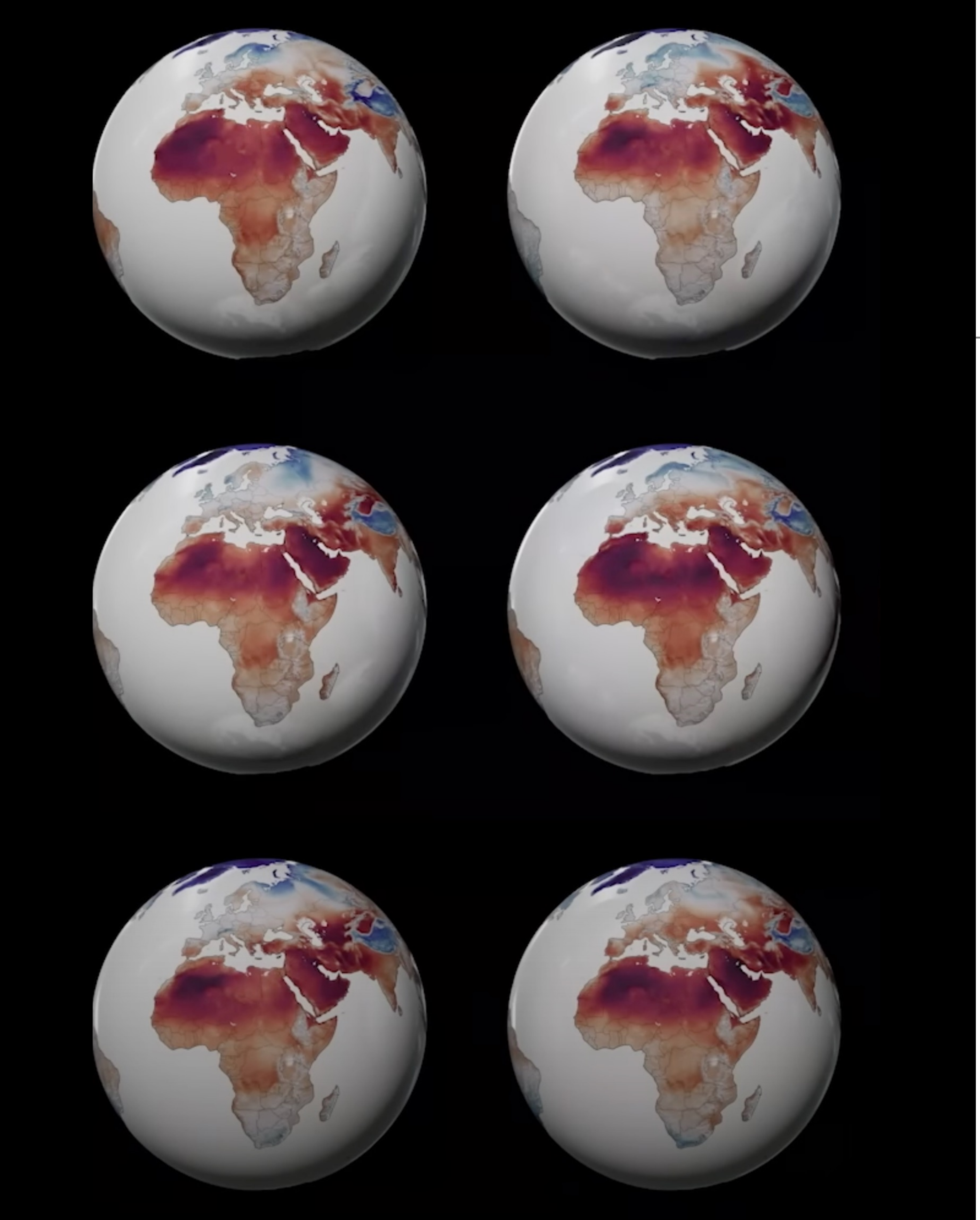


Predictions remain remarkably stable, even past the predictability horizon of weather.



Conclusion

- Spherical Generalization of Fourier Neural Operators
  - grid-invariance, model can be evaluated on any grid*
  - model can be trained on one resolution and deployed at another resolution
  - Correct treatment of spherical geometry and inherent symmetries
- Long-term stability with stable roll-outs of up to a year
- Remarkably similar to traditional spectral methods



Outlook

- Integration into existing weather prediction pipelines
  - Continuous training of the models and integration with data assimilation pipelines
  - Neural Operators support unstructured data and can integrate data from multiple sources
- Higher resolution, scaling properties, downcasting
- Prior physical knowledge, interpretability
- Large Ensemble forecasting and predicting extreme weather events

Thank you

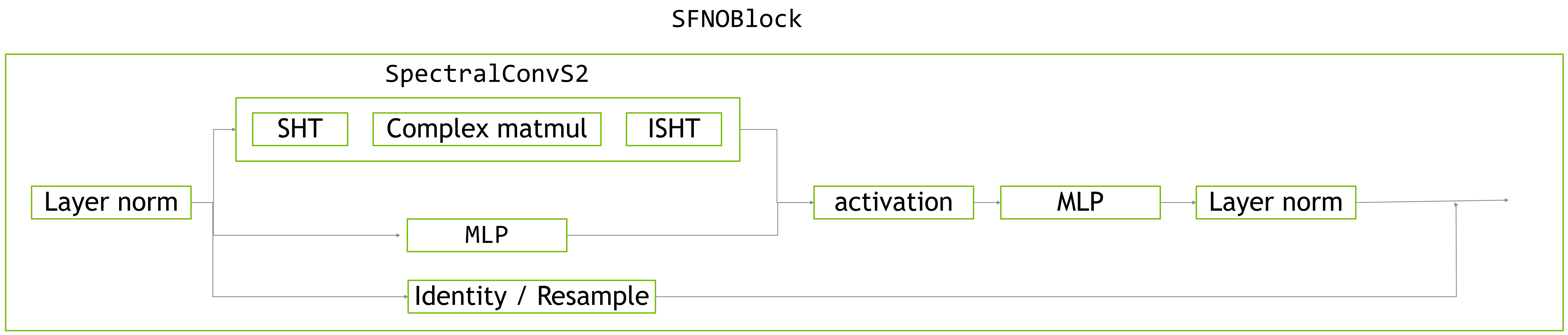
NVIDIA

Deep-dive

SFNO Deep Dive

Model overview

- Main building block of the model are the SFNO blocks
- In turn, these consist of a convolution, MLPs and layer norms similar to ConvNets
- Layer norms need to be formulated in an equivariant fashion -> InstanceNorm
- Global Convolution is achieved via SHT



SFNO Deep dive

Model overview

```python
class SphericalFourierNeuralOperatorBlock(nn.Module):
    """
    Helper module for a single SFNO/FNO block. Can use both FFTs and SHTs to represent either FNO or SFNO blocks.
    """
    def __init__(
            self,
            forward_transform,
            inverse_transform,
            embed_dim,
            filter_type = 'non-linear',
            operator_type = 'diagonal',
            mlp_ratio = 2.,
            drop_rate = 0.,
            drop_path = 0.,
            act_layer = nn.GELU,
            norm_layer = (nn.LayerNorm, nn.LayerNorm),
            sparsity_threshold = 0.0,
            use_complex_kernels = True,
            factorization = None,
            separable = False,
            rank = 128,
            inner_skip = 'linear',
            outer_skip = None, # None, nn.linear or nn.Identity
            concat_skip = False,
            use_mlp = True,
            complex_activation = 'real',
            spectral_layers = 3):
```

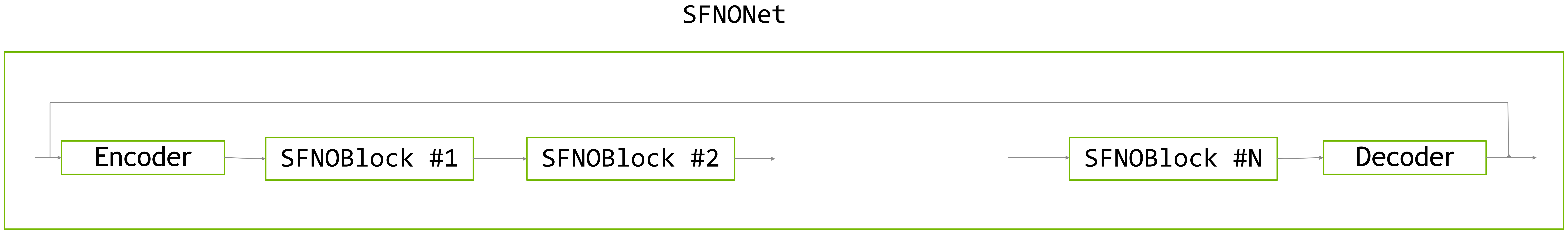
```python
class SpectralConvS2(nn.Module):
    """
    Spectral Convolution according to Driscoll & Healy. Designed for convolutions on the two-sphere S2
    using the Spherical Harmonic Transforms in torch-harmonics, but supports convolutions on the periodic
    domain via the RealFFT2 and InverseRealFFT2 wrappers.
    """

    def __init__(self,
                 forward_transform,
                 inverse_transform,
                 in_channels,
                 out_channels,
                 scale = 'auto',
                 operator_type = 'diagonal',
                 rank = 0.2,
                 factorization = None,
                 separable = False,
                 implementation = 'factorized',
                 decomposition_kwargs=dict(),
                 bias = False):
        super(SpectralConvS2, self).__init__()
```

Implementation in SpectralFourierNeuralOperatorBlock and SpectralConvS2

SFNO Deep Dive

Model overview

- Overall architecture build from SFNOBlocks
- Encoder and Decoder map the data to feature space and back
- Entire network is made up of either point-wise operations or global convolutions, thus retaining equivariance
- Model-parallelism “almost for free”. All you need is a parallel SHT and some logic to shard the model.
- A large skip connection is used as mapping is close to identity:
- As all operations are decoupled from the underlying Mesh, we have trained a **Neural Operator**
- This allows the model to be applied at different resolutions, and meshes, as long as we can formulate an SHT on that mesh



Spatial parallelism in SFNO

Achieving model parallelism via distributed SHT (torch-harmonics)

Training SFNO end-to-end

- End-to-end training example on the Spherical Shallow Water Equations on the rotating Sphere
- End-to-end training example available in notebook here: https://github.com/NVIDIA/torch-harmonics/blob/main/notebooks/train_sfno.ipynb
- For weather, we trained our models on the ERA5 dataset.
- Modulus has an implementation of SFNO: https://github.com/NVIDIA/modulus/tree/main/modulus/datapipes/climate

```python
# dataset
from torch_harmonics.examples.sfno import PdeDataset

# 1 hour prediction steps
dt = 1*3600
dt_solver = 150
nsteps = dt//dt_solver
dataset = PdeDataset(dt=dt, nsteps=nsteps, dims=(256, 512), device=device, normalize=True)
# There is still an issue with parallel dataloading. Do NOT use it at the moment
dataloader = DataLoader(dataset, batch_size=4, shuffle=True, num_workers=0, persistent_workers=False)
solver = dataset.solver.to(device)

nlat = dataset.nlat
nlon = dataset.nlon
```

[3] Python

```python
from torch_harmonics.examples.sfno import SphericalFourierNeuralOperatorNet as SFNO
```

[5] Python

```python
fno_model = SFNO(spectral_transform='sht', filter_type='linear', operator_type='vector', img_size=(nlat, nlon),
                 num_layers=4, scale_factor=3, embed_dim=256).to(device)
```

[6] Python

+ Code + Markdown

```
--------------------------------------------------------------------------------
Epoch 0 summary:
time taken: 30.337862968444824
accumulated training loss: 6.563157506287098
relative validation loss: 0.14439213275909424
--------------------------------------------------------------------------------
Epoch 1 summary:
time taken: 30.065420627593994
accumulated training loss: 1.0783583740703762
relative validation loss: 0.03158371150493622
--------------------------------------------------------------------------------
Epoch 2 summary:
time taken: 30.11690044403076
accumulated training loss: 0.5136246508918703
relative validation loss: 0.027934805490076542
--------------------------------------------------------------------------------
Epoch 3 summary:
time taken: 30.133581399917603
accumulated training loss: 0.37776567693799734
relative validation loss: 0.024108433164656162
--------------------------------------------------------------------------------
Epoch 4 summary:
time taken: 30.13297390937805
accumulated training loss: 0.36006640107370913
relative validation loss: 0.014237499330192804
...
accumulated training loss: 0.31660769623704255
relative validation loss: 0.017148463986814022
--------------------------------------------------------------------------------
```

Additional Material

Learn more about SFNO and NVIDIA's Earth-2 Initiative

To see how SFNO was used to generate thousands of ensemble members and predict the 2018 Algerian heat wave, watch this demo:
https://www.youtube.com/watch?v=FUUT6IrQjo4

If you want to learn more about SFNO, here are some additional resources:

- Read the paper: https://arxiv.org/pdf/2306.03838.pdf
- torch-harmonics: https://github.com/NVIDIA/torch-harmonics
- Implementation of SFNO: https://github.com/NVIDIA/torch-harmonics/blob/main/torch_harmonics/examples/sfno/models/sfno.py
- Getting started with torch-harmonics: https://github.com/NVIDIA/torch-harmonics/blob/main/notebooks/getting_started.ipynb
- Training SFNO on the Spherical Shallow Water Equations: https://github.com/NVIDIA/torch-harmonics/blob/main/notebooks/train_sfno.ipynb
- The SFNO is available, along with other architecture in neuraloperator: https://github.com/neuraloperator/neuraloperator
- Training SFNO on Shallow Water Equations in neuraloperator: https://neuraloperator.github.io/neuraloperator/dev/auto_examples/plot_SFNO_swe.html#sphx-glr-auto-examples-plot-sfno-swe-py
- Implementation of SFNO in Modulus: https://github.com/NVIDIA/modulus/blob/9640510e2064312ba523a4a06f38923eb977f6aa/modulus/models/sfno/sfnonet.py

Follow the links below to learn more about NVIDIA's Earth-2 initiative of creating a digital twin of Earth's atmosphere:

- https://blogs.nvidia.com/blog/2023/07/03/climate-research-next-wave/
- https://blogs.nvidia.com/blog/2021/11/12/earth-2-supercomputer/
- https://www.nvidia.com/en-us/high-performance-computing/earth-2/
- Watch Jensen's keynote at the Berlin summit for EVE: https://www.youtube.com/watch?v=GTJVpPsSwpI