Spherical Fourier Neural Operators

A simple notebook to showcase spherical Fourier Neural Operators

Preparation

In [1]:
```
import paddle
import paddle.nn as nn
from paddle.io import DataLoader
from paddle import amp
from paddle.optimizer.lr import OneCycleLR

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

from math import ceil, sqrt

import time

cmap='twilight_shifted'
```

```
/usr/local/lib/python3.10/dist-packages/paddle/utils/cpp_extension/extensi
on_utils.py:686: UserWarning: No ccache found. Please be aware that recomp
iling all source files may be required. You can download and install ccach
e from: https://github.com/ccache/ccache/blob/master/doc/INSTALL.md
  warnings.warn(warning_message)
```

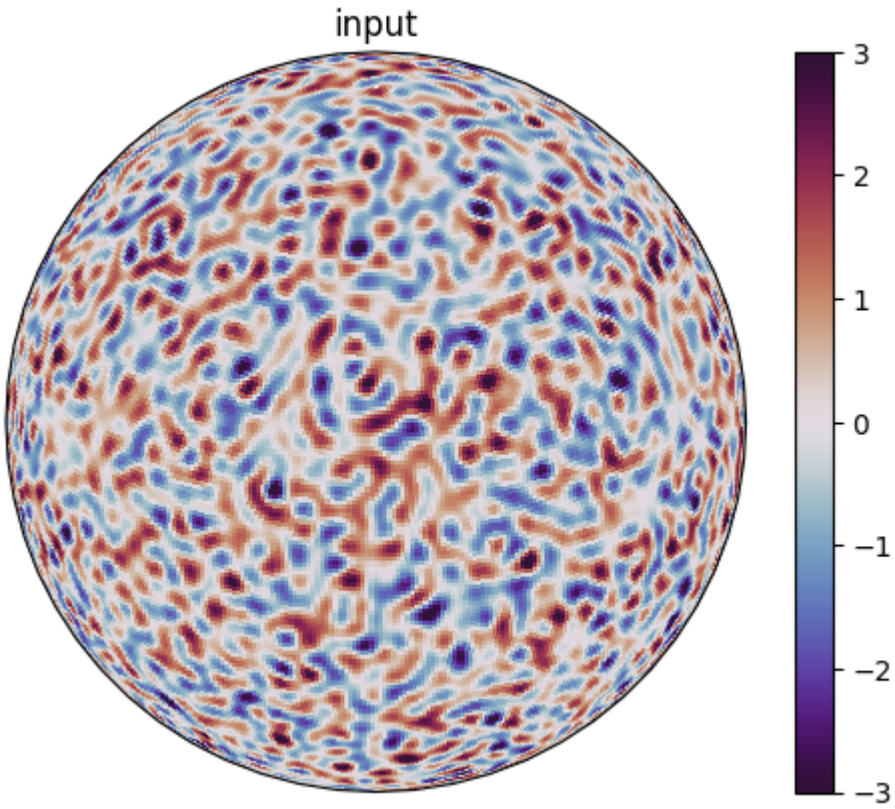
In [2]:
```
enable_amp = False

# set device
device = paddle.device.set_device('gpu' if paddle.device.cuda.device_coun
```

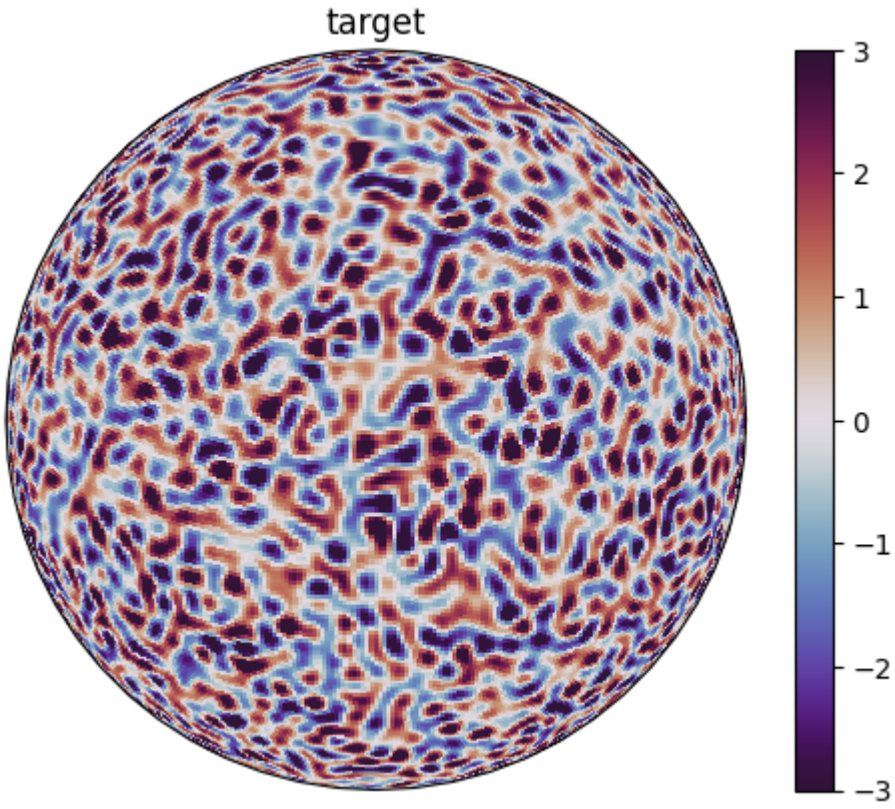
Training data

to train our geometric FNOs, we require training data. To this end let us prepare a Dataloader which computes results on the fly:

In [5]:
```
# dataset
from paddle_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
# There is still an issue with parallel dataloading. Do NOT use it at the
dataloader = DataLoader(dataset, batch_size=4, shuffle=True, num_workers=
solver = dataset.solver.to(device)

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

In [6]:
```
paddle.seed(0)
inp, tar = dataset[0]

fig = plt.figure()
im = solver.plot_griddata(inp[2], fig, vmax=3, vmin=-3)
plt.title("input")
plt.colorbar(im)
plt.show()

fig = plt.figure()
im = solver.plot_griddata(tar[2], fig, vmax=3, vmin=-3)
plt.title("target")
plt.colorbar(im)
plt.show()
```





Defining the geometric Fourier Neural Operator

In [7]:
```
from paddle_harmonics.examples.sfno import SphericalFourierNeuralOperator
```

In [8]:
```
model = SFNO(spectral_transform='sht', operator_type='driscoll-healy', im
                num_layers=4, scale_factor=3, embed_dim=16, big_skip=Tru
```

Training the model

In [9]:
```
def l2loss_sphere(solver, prd, tar, relative=False, squared=True):
    loss = solver.integrate_grid((prd - tar)**2, dimensionless=True).sum(
    if relative:
        loss = loss / solver.integrate_grid(tar**2, dimensionless=True).s

    if not squared:
        loss = paddle.sqrt(loss)
    loss = loss.mean()

    return loss

def spectral_l2loss_sphere(solver, prd, tar, relative=False, squared=True
    # compute coefficients
    coeffs = paddle.as_real(solver.sht(prd - tar))
    coeffs = coeffs[..., 0]**2 + coeffs[..., 1]**2
    norm2 = coeffs[..., :, 0] + 2 * paddle.sum(coeffs[..., :, 1:], axis=-
    loss = paddle.sum(norm2, axis=(-1,-2))

    if relative:
        tar_coeffs = paddle.as_real(solver.sht(tar))
        tar_coeffs = tar_coeffs[..., 0]**2 + tar_coeffs[..., 1]**2
        tar_norm2 = tar_coeffs[..., :, 0] + 2 * paddle.sum(tar_coeffs[...
```

```
        tar_norm2 = paddle.sum(tar_norm2, axis=(-1,-2))
        loss = loss / tar_norm2

    if not squared:
        loss = paddle.sqrt(loss)
    loss = loss.mean()

    return loss
```

In [10]:

```
# training function
def train_model(model, dataloader, optimizer, scheduler=None, nepochs=20,

    train_start = time.time()

    for epoch in range(nepochs):

        # time each epoch
        epoch_start = time.time()

        dataloader.dataset.set_initial_condition('random')
        dataloader.dataset.set_num_examples(num_examples)

        optimizer.clear_grad(set_to_zero=True)

        # do the training
        acc_loss = 0
        model.train()

        for inp, tar in dataloader:
            with amp.auto_cast(enable=enable_amp):
                prd = model(inp)
                for _ in range(nfuture):
                    prd = model(prd)
                if loss_fn == 'l2':
                    loss = l2loss_sphere(solver, prd, tar)
                elif loss_fn == "spectral l2":
                    loss = spectral_l2loss_sphere(solver, prd, tar)

            acc_loss += loss.item() * inp.shape[0]

            optimizer.clear_grad(set_to_zero=True)
            # gscaler.scale(loss).backward()
            loss.backward()
            optimizer.step()
            # gscaler.update()

        if scheduler is not None:
            scheduler.step()

        acc_loss = acc_loss / len(dataloader.dataset)

        dataloader.dataset.set_initial_condition('random')
        dataloader.dataset.set_num_examples(num_valid)

        # perform validation
        valid_loss = 0
        model.eval()
        with paddle.no_grad():
            for inp, tar in dataloader:
                prd = model(inp)
```

```
                for _ in range(nfuture):
                    prd = model(prd)
                loss = l2loss_sphere(solver, prd, tar, relative=True)

                valid_loss += loss.item() * inp.shape[0]

        valid_loss = valid_loss / len(dataloader.dataset)

        epoch_time = time.time() - epoch_start

        print(f'-------------------------------------------------------------
        print(f'Epoch {epoch} summary:')
        print(f'time taken: {epoch_time}')
        print(f'accumulated training loss: {acc_loss}')
        print(f'relative validation loss: {valid_loss}')

    train_time = time.time() - train_start

    print(f'-------------------------------------------------------------------
    print(f'done. Training took {train_time}.')
    return valid_loss
```

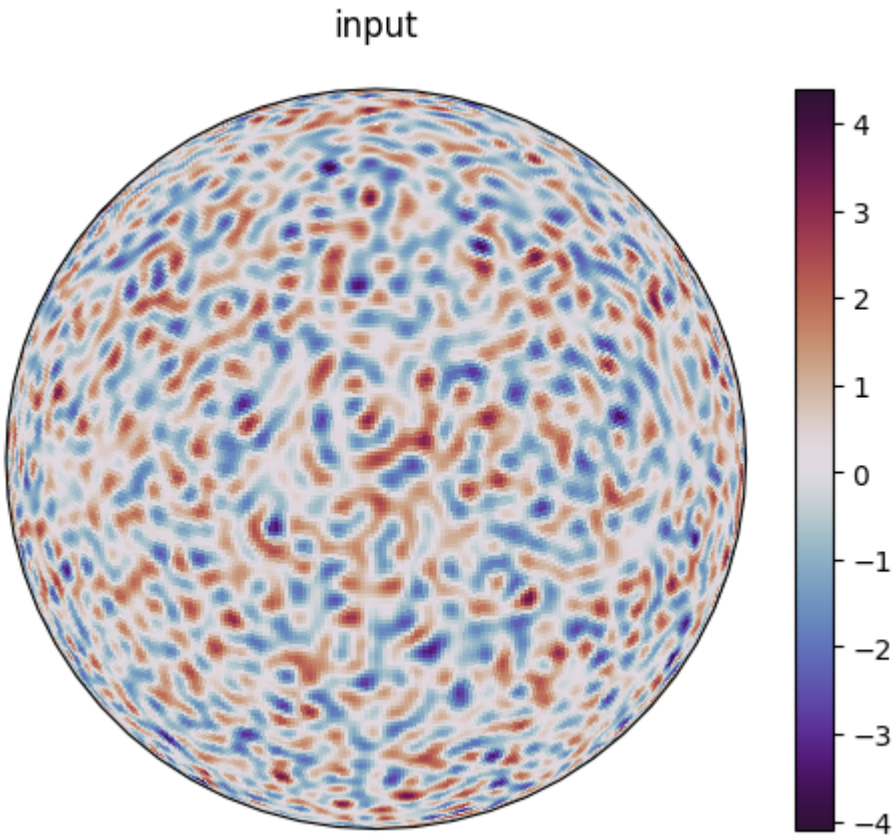
In [11]:
```
# set seed
paddle.seed(333)

optimizer = paddle.optimizer.Adam(parameters=model.parameters(), learning
gscaler = amp.GradScaler(enable=enable_amp)
train_model(model, dataloader, optimizer, nepochs=10)
```

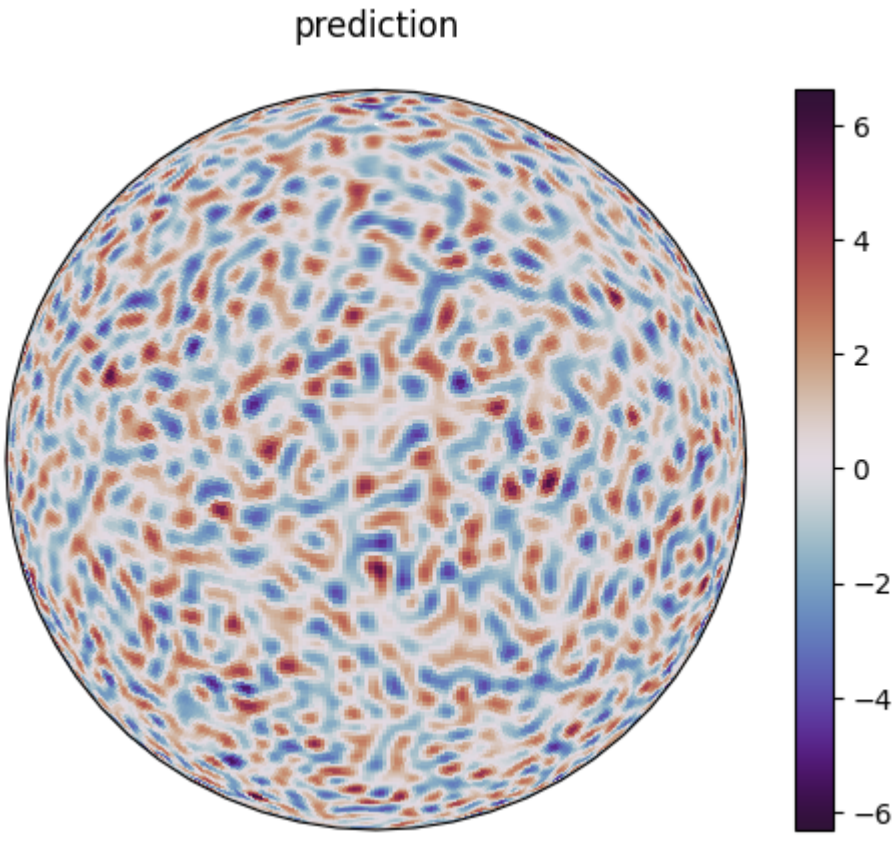
```
------------------------------------------------------------------------
------
Epoch 0 summary:
time taken: 58.65468692779541
accumulated training loss: 63.65380771458149
relative validation loss: 0.2742103785276413
------------------------------------------------------------------------
------
Epoch 1 summary:
time taken: 57.2250919342041
accumulated training loss: 10.009205937385559
relative validation loss: 0.1523861512541771
------------------------------------------------------------------------
------
Epoch 2 summary:
time taken: 57.95978879928589
accumulated training loss: 6.189137250185013
relative validation loss: 0.10285826399922371
------------------------------------------------------------------------
------
Epoch 3 summary:
time taken: 57.21123504638672
accumulated training loss: 4.3623782098293304
relative validation loss: 0.07523632049560547
------------------------------------------------------------------------
------
Epoch 4 summary:
time taken: 56.73054766654968
accumulated training loss: 3.326366003602743
relative validation loss: 0.05963883548974991
------------------------------------------------------------------------
------
Epoch 5 summary:
time taken: 58.39644646644592
accumulated training loss: 2.6672437824308872
relative validation loss: 0.04848321154713631
------------------------------------------------------------------------
------
Epoch 6 summary:
time taken: 57.704843044281006
accumulated training loss: 2.215153779834509
relative validation loss: 0.04129674844443798
------------------------------------------------------------------------
------
Epoch 7 summary:
time taken: 57.756645917892456
accumulated training loss: 1.867876697331667
relative validation loss: 0.035139940679073334
------------------------------------------------------------------------
------
Epoch 8 summary:
time taken: 56.731836557388306
accumulated training loss: 1.5758926887065172
relative validation loss: 0.029634888283908367
------------------------------------------------------------------------
------
Epoch 9 summary:
time taken: 57.77064394950867
accumulated training loss: 1.3386675417423248
relative validation loss: 0.024751360528171062
```

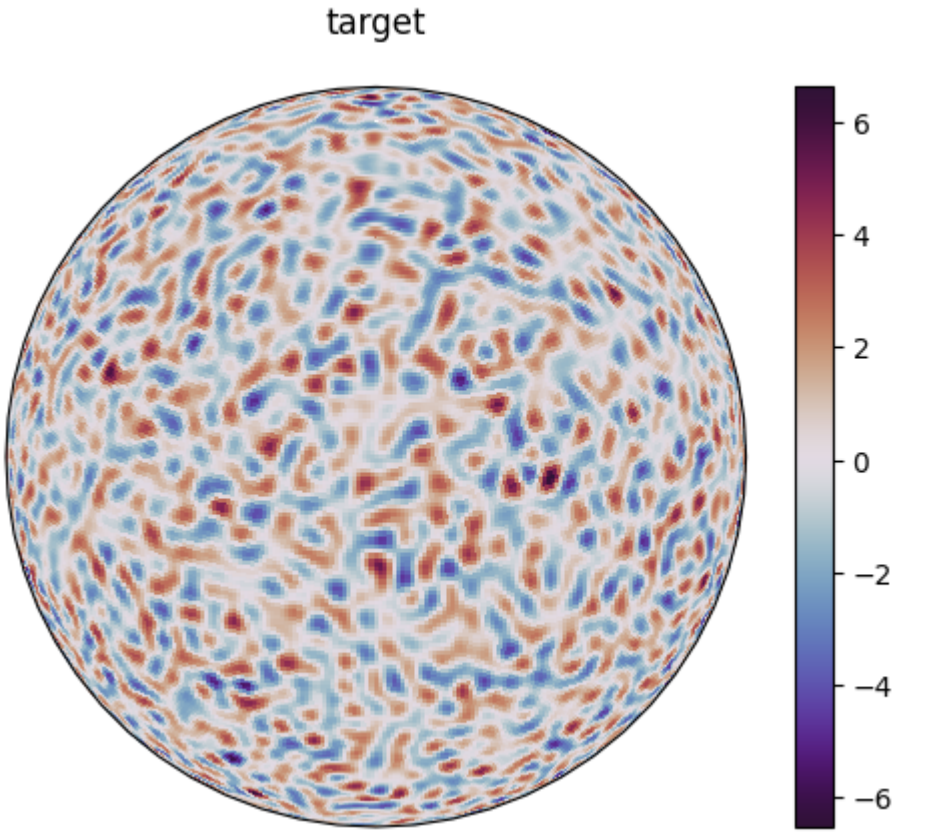
```
------------------------------------------------------------------------
------
done. Training took 576.1440336704254.
```

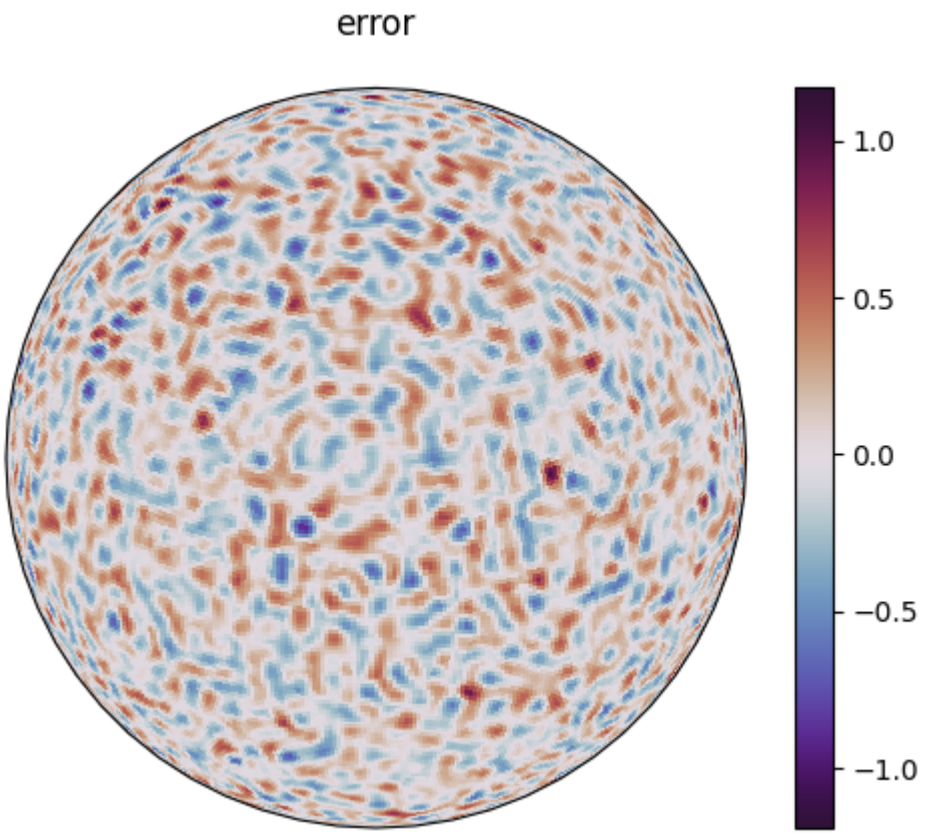
Out[11]: 0.024751360528171062

In [12]:
```
dataloader.dataset.set_initial_condition('random')

paddle.seed(0)

with paddle.no_grad():
    inp, tar = next(iter(dataloader))
    out = model(inp).detach()

s = 0; ch = 2

fig = plt.figure()
im = solver.plot_griddata(inp[s, ch], fig, projection='3d', title='input'
plt.colorbar(im)
plt.show()

fig = plt.figure()
im = solver.plot_griddata(out[s, ch], fig, projection='3d', title='predic
plt.colorbar(im)
plt.show()

fig = plt.figure()
im = solver.plot_griddata(tar[s, ch], fig, projection='3d', title='target
plt.colorbar(im)
plt.show()

fig = plt.figure()
im = solver.plot_griddata((tar-out)[s, ch], fig, projection='3d', title='
plt.colorbar(im)
plt.show()
```


input
4
3
2
1
0
−1
−2
−3
−4


prediction
6
4
2
0
−2
−4
−6





In [ ]: