In [1]:
```
## Import modules

from Fourier_Neural_Operator import Fourier_Neural_Operator as FNO
from Fourier_Neural_Operator import SpectralConv2d_fast, SimpleBlock2d, Net2d
from mpl_toolkits.axes_grid1 import ImageGrid
import matplotlib.pyplot as plt
import numpy as np
import torch
```

In [26]:
```
## Plot function tests

# Simple ImageGrid plot for all 10 timesteps

def plot_images(data1):
    fig = plt.figure(figsize=(40., 40.))
    grid = ImageGrid(fig, 111,  # similar to subplot(111)
                     nrows_ncols=(1, 10),  # creates 2x2 grid of axes
                     axes_pad=0.1,  # pad between axes in inch.
                     )
    for i, ax in zip(range(0, 10),grid):
        # Iterating over the grid returns the Axes. #shape of arr is (nx, ny, time, channe
        ax.imshow(np.array(data1.detach())[1,:,:,0,i])
        ax.set_title("Timestep: '{0}'".format(i))

# Plot function to compare prediction and ground truth

def plot_image_compare(pred, truth):
    fig = plt.figure(figsize=(40., 40.))
    grid = ImageGrid(fig, 111,  # similar to subplot(111)
                     nrows_ncols=(2, 3),  # creates 2x2 grid of axes
                     axes_pad=1,  # pad between axes in inch.
                     )
    im1 = np.array(truth.detach())[1,:,:,1,0]
    im2 = np.array(truth.detach())[1,:,:,1,5]
    im3 = np.array(truth.detach())[1,:,:,1,9]
    im4 = np.array(pred.detach())[1,:,:,1,0]
    im5 = np.array(pred.detach())[1,:,:,1,5]
    im6 = np.array(pred.detach())[1,:,:,1,9]
    for ax, im in zip(grid, [im1, im2, im3, im4, im5, im6]):
        ax.imshow(im)
        ax.set_title("")

## Plot function to compare prediction and ground truth

# Function to add all channels in a single image
def stack_channels(images):
    c_a = np.array(images.detach())[:,:,:,0,:]
    c_b = np.array(images.detach())[:,:,:,1,:]
    c_c = 1 - c_a - c_b
    image = np.stack((c_a, c_b, c_c), axis = 3)
    return image

def plot_image_compare_sp(pred, truth, fname = 'None'):
    f, axes = plt.subplots(2, 3, figsize=(10,8))

    pred_im = stack_channels(pred) # Stack channels function
    truth_im = stack_channels(truth) # Stack channels function

    pred = pred_im[1,:,:,:,:]
    truth = truth_im[1,:,:,:,:]

    print(pred.shape, truth.shape)
```

```
    f.subplots_adjust(hspace = 0.5)

    for ax, i in zip(axes[0,:], range(1, len(truth[0,0,0,:]), 4)):
        ax.imshow(pred[:,:,:,i])
        # i += 4
        ax.set_title('Prediction - Timestep {}'.format(i))
    for ax, i in zip(axes[1,:], range(1, len(pred[0,0,0,:]), 4)):
        ax.imshow(truth[:,:,:,i])
        # i += 4
        ax.set_title('Ground Truth - Timestep {}'.format(i))

    plt.tight_layout()
    plt.savefig(fname)
```

In [28]:
```
## Upload model and dataset

import torch

checkpoint = torch.load('model_ts5.pt', map_location=torch.device('cpu'))
model = Net2d(12, 10) # (modes, width)
model.load_state_dict(checkpoint['model_state_dict'])

D = np.load('Data_dt5_time_jump.npy')
```

In [29]:
```
## Data organiaztion

sub = 1
S = 64
T_in = 10
T = 100
step = 1

ntest = 2

batch_size = 20

test_a = torch.tensor(D[-ntest:,::sub,::sub,::sub,:T_in])
test_u = torch.tensor(D[-ntest:,::sub,::sub,::sub,T_in:T+T_in])

gridx = torch.tensor(np.linspace(0, 64, S), dtype=torch.float)
gridx = gridx.reshape(1, S, 1, 1).repeat([1, 1, S, 1])
gridx = gridx.reshape(1, S, S, 1, 1).repeat([1, 1, 1, 2, 1])
gridy = torch.tensor(np.linspace(0, 64, S), dtype=torch.float)
gridy = gridy.reshape(1, 1, S, 1).repeat([1, S, 1, 1])
gridy = gridy.reshape(1, S, S, 1, 1).repeat([1, 1, 1, 2, 1])

test_a = torch.cat((test_a, gridx.repeat([ntest,1,1,1,1]), gridy.repeat([ntest,1,1,1,1])),

test_loader = torch.utils.data.DataLoader(torch.utils.data.TensorDataset(test_a, test_u),
```

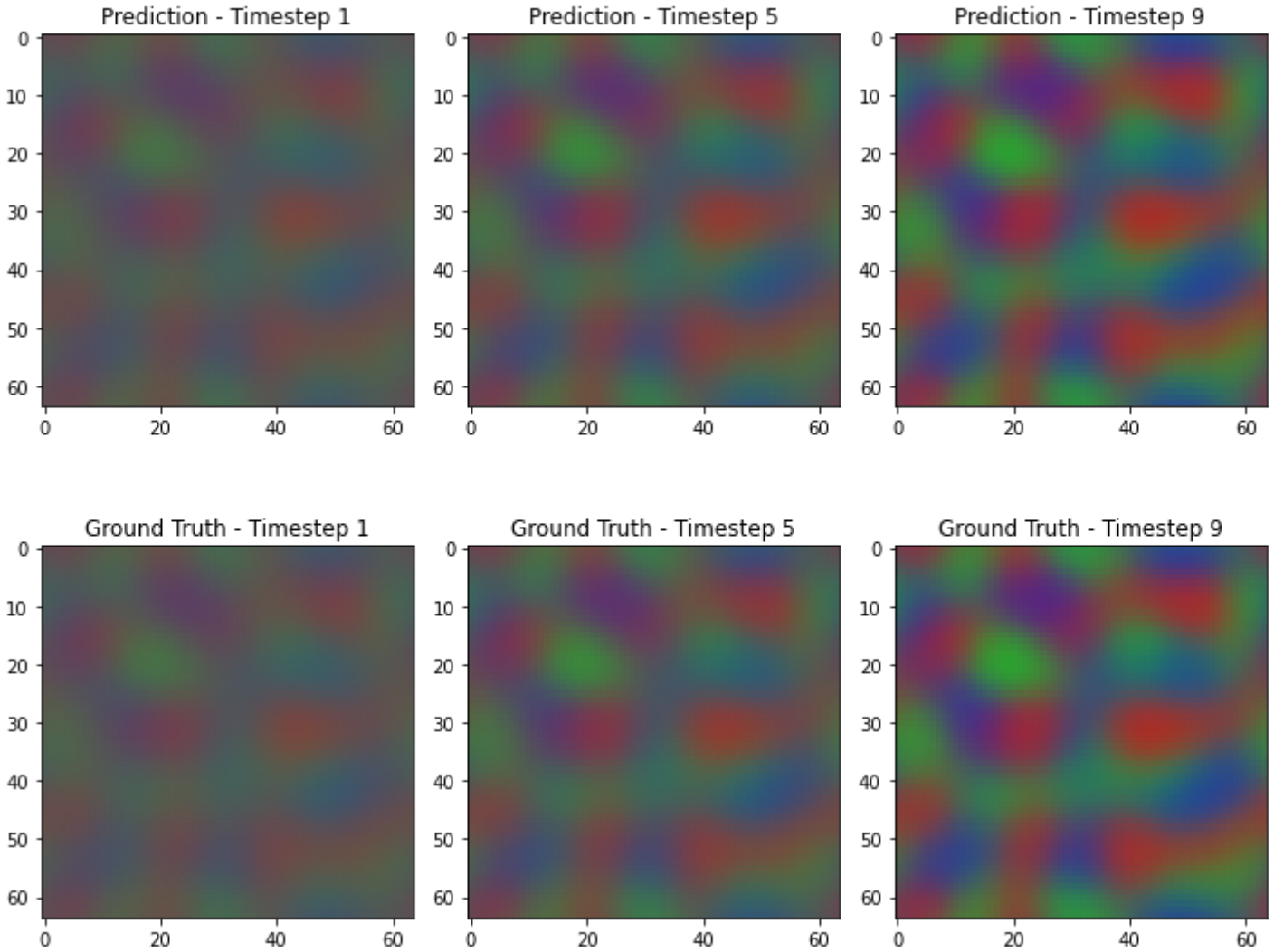
In [30]:
```
## Predictions
xx = []
yy = []
yh = []
for x, y in test_loader:
    xx.append(x)
    yy.append(y)

x = xx[0]

for t in range(0, T+10, step):
```

```
    im = model(x.float())

    if t ==0:
        pred = im
    else:
        pred = torch.cat((pred, im), -1)
        # pred = im

    x = torch.cat((x[..., step:-2], im, gridx.repeat([ntest, 1, 1, 1, 1]), gridy.repeat([r
```

In [31]:

```
## Compare images predictions vd truth

plot_image_compare_sp(pred, yy[0], fname = '256.png')
```

```
(64, 64, 3, 110) (64, 64, 3, 100)
```



In [32]:

```
## Predictions till first 100 steps and graph

from torch import nn

loss = []
for i in range(0,100):
    ls = nn.MSELoss(size_average = True, reduce = True, reduction = 'mean')(pred[1,:,:,1,i
    print(i, ls)
    loss.append(ls)

plt.plot(np.array(loss))
plt.xticks(np.arange(0, 100, 10))
plt.savefig('LossCurve_dt5_w10.jpg')
```

```
C:\Users\A Maruthi Indresh\anaconda3\lib\site-packages\torch\nn\_reduction.py:42: UserWarning: size_average and reduce args will be deprecated, please use reduction='mean' instead.
  warnings.warn(warning.format(ret))
0 tensor(2.2873e-07, dtype=torch.float64, grad_fn=<MseLossBackward>)
1 tensor(2.0765e-07, dtype=torch.float64, grad_fn=<MseLossBackward>)
2 tensor(3.1900e-07, dtype=torch.float64, grad_fn=<MseLossBackward>)
3 tensor(5.9188e-07, dtype=torch.float64, grad_fn=<MseLossBackward>)
4 tensor(1.0986e-06, dtype=torch.float64, grad_fn=<MseLossBackward>)
5 tensor(2.0200e-06, dtype=torch.float64, grad_fn=<MseLossBackward>)
6 tensor(3.9483e-06, dtype=torch.float64, grad_fn=<MseLossBackward>)
7 tensor(7.8186e-06, dtype=torch.float64, grad_fn=<MseLossBackward>)
8 tensor(1.4661e-05, dtype=torch.float64, grad_fn=<MseLossBackward>)
9 tensor(2.5869e-05, dtype=torch.float64, grad_fn=<MseLossBackward>)
10 tensor(4.7966e-05, dtype=torch.float64, grad_fn=<MseLossBackward>)
11 tensor(9.9809e-05, dtype=torch.float64, grad_fn=<MseLossBackward>)
12 tensor(0.0002, dtype=torch.float64, grad_fn=<MseLossBackward>)
13 tensor(0.0005, dtype=torch.float64, grad_fn=<MseLossBackward>)
14 tensor(0.0010, dtype=torch.float64, grad_fn=<MseLossBackward>)
15 tensor(0.0020, dtype=torch.float64, grad_fn=<MseLossBackward>)
16 tensor(0.0036, dtype=torch.float64, grad_fn=<MseLossBackward>)
17 tensor(0.0060, dtype=torch.float64, grad_fn=<MseLossBackward>)
18 tensor(0.0092, dtype=torch.float64, grad_fn=<MseLossBackward>)
19 tensor(0.0132, dtype=torch.float64, grad_fn=<MseLossBackward>)
20 tensor(0.0177, dtype=torch.float64, grad_fn=<MseLossBackward>)
21 tensor(0.0226, dtype=torch.float64, grad_fn=<MseLossBackward>)
22 tensor(0.0271, dtype=torch.float64, grad_fn=<MseLossBackward>)
23 tensor(0.0313, dtype=torch.float64, grad_fn=<MseLossBackward>)
24 tensor(0.0349, dtype=torch.float64, grad_fn=<MseLossBackward>)
25 tensor(0.0382, dtype=torch.float64, grad_fn=<MseLossBackward>)
26 tensor(0.0413, dtype=torch.float64, grad_fn=<MseLossBackward>)
27 tensor(0.0440, dtype=torch.float64, grad_fn=<MseLossBackward>)
28 tensor(0.0461, dtype=torch.float64, grad_fn=<MseLossBackward>)
29 tensor(0.0479, dtype=torch.float64, grad_fn=<MseLossBackward>)
30 tensor(0.0494, dtype=torch.float64, grad_fn=<MseLossBackward>)
31 tensor(0.0504, dtype=torch.float64, grad_fn=<MseLossBackward>)
32 tensor(0.0512, dtype=torch.float64, grad_fn=<MseLossBackward>)
33 tensor(0.0519, dtype=torch.float64, grad_fn=<MseLossBackward>)
34 tensor(0.0526, dtype=torch.float64, grad_fn=<MseLossBackward>)
35 tensor(0.0532, dtype=torch.float64, grad_fn=<MseLossBackward>)
36 tensor(0.0538, dtype=torch.float64, grad_fn=<MseLossBackward>)
37 tensor(0.0543, dtype=torch.float64, grad_fn=<MseLossBackward>)
38 tensor(0.0548, dtype=torch.float64, grad_fn=<MseLossBackward>)
39 tensor(0.0552, dtype=torch.float64, grad_fn=<MseLossBackward>)
40 tensor(0.0556, dtype=torch.float64, grad_fn=<MseLossBackward>)
41 tensor(0.0560, dtype=torch.float64, grad_fn=<MseLossBackward>)
42 tensor(0.0566, dtype=torch.float64, grad_fn=<MseLossBackward>)
43 tensor(0.0572, dtype=torch.float64, grad_fn=<MseLossBackward>)
44 tensor(0.0578, dtype=torch.float64, grad_fn=<MseLossBackward>)
45 tensor(0.0584, dtype=torch.float64, grad_fn=<MseLossBackward>)
46 tensor(0.0589, dtype=torch.float64, grad_fn=<MseLossBackward>)
47 tensor(0.0594, dtype=torch.float64, grad_fn=<MseLossBackward>)
48 tensor(0.0598, dtype=torch.float64, grad_fn=<MseLossBackward>)
49 tensor(0.0601, dtype=torch.float64, grad_fn=<MseLossBackward>)
50 tensor(0.0604, dtype=torch.float64, grad_fn=<MseLossBackward>)
51 tensor(0.0605, dtype=torch.float64, grad_fn=<MseLossBackward>)
52 tensor(0.0608, dtype=torch.float64, grad_fn=<MseLossBackward>)
53 tensor(0.0611, dtype=torch.float64, grad_fn=<MseLossBackward>)
54 tensor(0.0616, dtype=torch.float64, grad_fn=<MseLossBackward>)
55 tensor(0.0621, dtype=torch.float64, grad_fn=<MseLossBackward>)
56 tensor(0.0628, dtype=torch.float64, grad_fn=<MseLossBackward>)
57 tensor(0.0634, dtype=torch.float64, grad_fn=<MseLossBackward>)
58 tensor(0.0639, dtype=torch.float64, grad_fn=<MseLossBackward>)
59 tensor(0.0643, dtype=torch.float64, grad_fn=<MseLossBackward>)
60 tensor(0.0648, dtype=torch.float64, grad_fn=<MseLossBackward>)
61 tensor(0.0652, dtype=torch.float64, grad_fn=<MseLossBackward>)
62 tensor(0.0655, dtype=torch.float64, grad_fn=<MseLossBackward>)
63 tensor(0.0659, dtype=torch.float64, grad_fn=<MseLossBackward>)
64 tensor(0.0661, dtype=torch.float64, grad_fn=<MseLossBackward>)
65 tensor(0.0665, dtype=torch.float64, grad_fn=<MseLossBackward>)
68, dtype=torch.float64, grad_fn=<MseLossBackward>)
```

```
67 tensor(0.0672, dtype=torch.float64, grad_fn=<MseLossBackward>)
68 tensor(0.0677, dtype=torch.float64, grad_fn=<MseLossBackward>)
69 tensor(0.0680, dtype=torch.float64, grad_fn=<MseLossBackward>)
70 tensor(0.0682, dtype=torch.float64, grad_fn=<MseLossBackward>)
71 tensor(0.0685, dtype=torch.float64, grad_fn=<MseLossBackward>)
72 tensor(0.0689, dtype=torch.float64, grad_fn=<MseLossBackward>)
73 tensor(0.0693, dtype=torch.float64, grad_fn=<MseLossBackward>)
74 tensor(0.0699, dtype=torch.float64, grad_fn=<MseLossBackward>)
75 tensor(0.0705, dtype=torch.float64, grad_fn=<MseLossBackward>)
76 tensor(0.0710, dtype=torch.float64, grad_fn=<MseLossBackward>)
77 tensor(0.0715, dtype=torch.float64, grad_fn=<MseLossBackward>)
78 tensor(0.0719, dtype=torch.float64, grad_fn=<MseLossBackward>)
79 tensor(0.0721, dtype=torch.float64, grad_fn=<MseLossBackward>)
80 tensor(0.0722, dtype=torch.float64, grad_fn=<MseLossBackward>)
81 tensor(0.0722, dtype=torch.float64, grad_fn=<MseLossBackward>)
82 tensor(0.0723, dtype=torch.float64, grad_fn=<MseLossBackward>)
83 tensor(0.0724, dtype=torch.float64, grad_fn=<MseLossBackward>)
84 tensor(0.0726, dtype=torch.float64, grad_fn=<MseLossBackward>)
85 tensor(0.0728, dtype=torch.float64, grad_fn=<MseLossBackward>)
86 tensor(0.0731, dtype=torch.float64, grad_fn=<MseLossBackward>)
87 tensor(0.0734, dtype=torch.float64, grad_fn=<MseLossBackward>)
88 tensor(0.0737, dtype=torch.float64, grad_fn=<MseLossBackward>)
89 tensor(0.0739, dtype=torch.float64, grad_fn=<MseLossBackward>)
90 tensor(0.0741, dtype=torch.float64, grad_fn=<MseLossBackward>)
91 tensor(0.0743, dtype=torch.float64, grad_fn=<MseLossBackward>)
92 tensor(0.0745, dtype=torch.float64, grad_fn=<MseLossBackward>)
93 tensor(0.0748, dtype=torch.float64, grad_fn=<MseLossBackward>)
94 tensor(0.0750, dtype=torch.float64, grad_fn=<MseLossBackward>)
95 tensor(0.0752, dtype=torch.float64, grad_fn=<MseLossBackward>)
96 tensor(0.0754, dtype=torch.float64, grad_fn=<MseLossBackward>)
97 tensor(0.0757, dtype=torch.float64, grad_fn=<MseLossBackward>)
98 tensor(0.0758, dtype=torch.float64, grad_fn=<MseLossBackward>)
99 tensor(0.0760, dtype=torch.float64, grad_fn=<MseLossBackward>)
```