```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

```
!pip install cartopy
```

```
Collecting cartopy
  Downloading Cartopy-0.24.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (7.9 kB)
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/python3.11/dist-packages (from cartopy) (2.0.2)
Requirement already satisfied: matplotlib>=3.6 in /usr/local/lib/python3.11/dist-packages (from cartopy) (3.10.0)
Requirement already satisfied: shapely>=1.8 in /usr/local/lib/python3.11/dist-packages (from cartopy) (2.1.1)
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.11/dist-packages (from cartopy) (24.2)
Requirement already satisfied: pyshp>=2.3 in /usr/local/lib/python3.11/dist-packages (from cartopy) (2.3.1)
Requirement already satisfied: pyproj>=3.3.1 in /usr/local/lib/python3.11/dist-packages (from cartopy) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy) (
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy) (0.12
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy) (
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartop
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from pyproj>=3.3.1->cartopy) (2025.6.15)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib>=3
Downloading Cartopy-0.24.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.7/11.7 MB 122.4 MB/s eta 0:00:00
Installing collected packages: cartopy
Successfully installed cartopy-0.24.1
```

```
!ls drive/MyDrive/earth_engine/*.nc
```

```
drive/MyDrive/earth_engine/aod_avhrr.nc
drive/MyDrive/earth_engine/aod_modis.nc
drive/MyDrive/earth_engine/dynamic_world_built_2015_2024.nc
drive/MyDrive/earth_engine/dynamic_world_built_2015.nc
drive/MyDrive/earth_engine/dynamic_world_built_albuquerque.nc
drive/MyDrive/earth_engine/dynamic_world_built_austin.nc
drive/MyDrive/earth_engine/dynamic_world_built_columbus.nc
drive/MyDrive/earth_engine/dynamic_world_built_Delhi.nc
drive/MyDrive/earth_engine/dynamic_world_built_Dubai.nc
drive/MyDrive/earth_engine/dynamic_world_built_HongKong.nc
drive/MyDrive/earth_engine/dynamic_world_built_kansas_city.nc
drive/MyDrive/earth_engine/dynamic_world_built_Las_Vegas.nc
drive/MyDrive/earth_engine/dynamic_world_built_London.nc
drive/MyDrive/earth_engine/dynamic_world_built_Melbourne.nc
drive/MyDrive/earth_engine/dynamic_world_built_minneapolis.nc
drive/MyDrive/earth_engine/dynamic_world_built_portland.nc
drive/MyDrive/earth_engine/dynamic_world_built_seattle.nc
drive/MyDrive/earth_engine/dynamic_world_built_washington_DC.nc
drive/MyDrive/earth_engine/dynamic_world_flooded_vegetation_2015_2023.nc
drive/MyDrive/earth_engine/dynamic_world_water_2015_2023.nc
drive/MyDrive/earth_engine/noaa_aorc_pr_austin_2015.nc
drive/MyDrive/earth_engine/noaa_aorc_pr_austin_2016.nc
drive/MyDrive/earth_engine/noaa_aorc_pr_austin_2017.nc
drive/MyDrive/earth_engine/noaa_aorc_pr_austin_2018.nc
drive/MyDrive/earth_engine/noaa_aorc_pr_austin_2019.nc
drive/MyDrive/earth_engine/noaa_aorc_pr_austin_2020.nc
drive/MyDrive/earth_engine/noaa_aorc_pr_austin_2021.nc
drive/MyDrive/earth_engine/noaa_aorc_pr_austin_2022.nc
drive/MyDrive/earth_engine/noaa_aorc_pr_austin_2023.nc
drive/MyDrive/earth_engine/pr_era5_india.nc
drive/MyDrive/earth_engine/sm_era5_india.nc
drive/MyDrive/earth_engine/soil_texture_austin_ksat_mean.nc
drive/MyDrive/earth_engine/soil_texture_austin_theta_r_mean.nc
drive/MyDrive/earth_engine/soil_texture_austin_theta_s_mean.nc
```

```
import xarray as xr
!git clone https://github.com/manmeet3591/CYGNSS
```

```
Cloning into 'CYGNSS'...
remote: Enumerating objects: 432, done.
remote: Counting objects: 100% (166/166), done.
remote: Compressing objects: 100% (49/49), done.
remote: Total 432 (delta 134), reused 120 (delta 115), pack-reused 266 (from 1)
Receiving objects: 100% (432/432), 234.69 MiB | 61.17 MiB/s, done.
Resolving deltas: 100% (388/388), done.
Updating files: 100% (404/404), done.
```

```
import xarray as xr

vars = ['SM_daily', 'latitude', 'longitude']

ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???.dap.nc')[vars].compute()
ds_cygnss
```

```
/tmp/ipython-input-155-2464384813.py:5: FutureWarning: In a future version of
  ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???
/tmp/ipython-input-155-2464384813.py:5: FutureWarning: In a future version of
  ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???
/tmp/ipython-input-155-2464384813.py:5: FutureWarning: In a future version of
  ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???
/tmp/ipython-input-155-2464384813.py:5: FutureWarning: In a future version of
  ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???
/tmp/ipython-input-155-2464384813.py:5: FutureWarning: In a future version of
  ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???
/tmp/ipython-input-155-2464384813.py:5: FutureWarning: In a future version of
  ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???
/tmp/ipython-input-155-2464384813.py:5: FutureWarning: In a future version of
  ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???
/tmp/ipython-input-155-2464384813.py:5: FutureWarning: In a future version of
  ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???
/tmp/ipython-input-155-2464384813.py:5: FutureWarning: In a future version of
  ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???
/tmp/ipython-input-155-2464384813.py:5: FutureWarning: In a future version of
  ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???
/tmp/ipython-input-155-2464384813.py:5: FutureWarning: In a future version of
  ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???
/tmp/ipython-input-155-2464384813.py:5: FutureWarning: In a future version of
  ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???
/tmp/ipython-input-155-2464384813.py:5: FutureWarning: In a future version of
  ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???
/tmp/ipython-input-155-2464384813.py:5: FutureWarning: In a future version of
  ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???
/tmp/ipython-input-155-2464384813.py:5: FutureWarning: In a future version of
  ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???
/tmp/ipython-input-155-2464384813.py:5: FutureWarning: In a future version of
  ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???
/tmp/ipython-input-155-2464384813.py:5: FutureWarning: In a future version of
  ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???
/tmp/ipython-input-155-2464384813.py:5: FutureWarning: In a future version of
  ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???
/tmp/ipython-input-155-2464384813.py:5: FutureWarning: In a future version of
  ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???
/tmp/ipython-input-155-2464384813.py:5: FutureWarning: In a future version of
  ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???
```

xarray.Dataset

▸ Dimensions: (**time**: 360, lat: 252, lon: 802)

▾ Coordinates:

time	(time)	datetime64[ns]	2022-01-01T12:00:00 ... 2022-12-...

▾ Data variables:

SM_daily	(time, lat, lon)	float32	nan nan nan nan ... nan nan nan nan
latitude	(time, lat, lon)	float32	-38.14 -38.14 ... 38.14 38.14
longitude	(time, lat, lon)	float32	-135.0 -134.6 ... 163.8 164.1

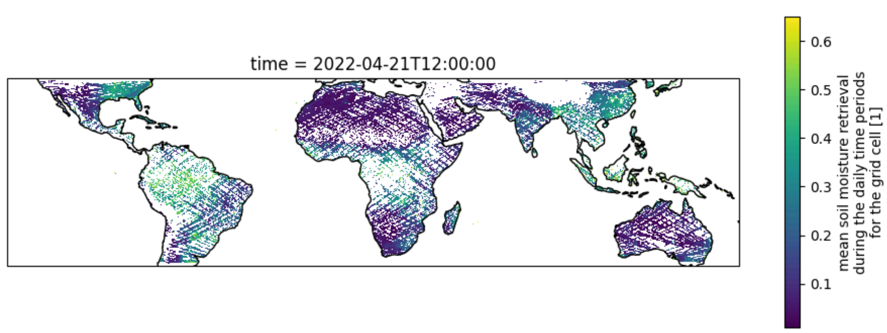
▸ Indexes: (1)

▸ Attributes: (45)

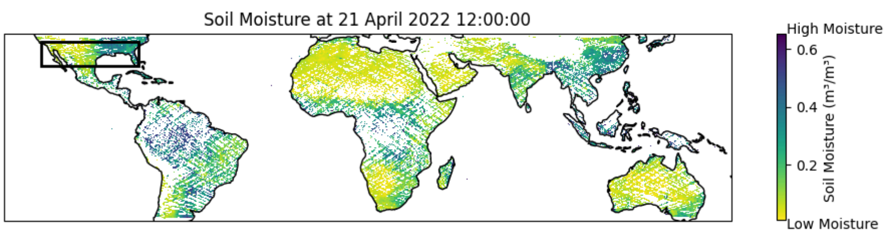
```python
# Adding latitude and longitude as 1D coordinates
lat_data = ds_cygnss['latitude'].isel(time=0, lon=0).values
lon_data = ds_cygnss['longitude'].isel(time=0, lat=0).values

ds_cygnss_ = ds_cygnss.assign_coords(lat=lat_data, lon=lon_data)
ds_cygnss_
```

```
xarray.Dataset
▸ Dimensions:        (time: 360, lat: 252, lon: 802)
▾ Coordinates:
    time             (time)            datetime64[ns]  2022-01-01T12:00:00 ... 2022-12-...
    lat              (lat)             float32         -38.14 -37.79 ... 37.79 38.14
    lon              (lon)             float32         -135.0 -134.6 ... 163.8 164.1
▾ Data variables:
    SM_daily         (time, lat, lon)  float32         nan nan nan nan ... nan nan nan nan
    latitude         (time, lat, lon)  float32         -38.14 -38.14 ... 38.14 38.14
    longitude        (time, lat, lon)  float32         -135.0 -134.6 ... 163.8 164.1
▸ Indexes:   (3)
▸ Attributes:   (45)
```

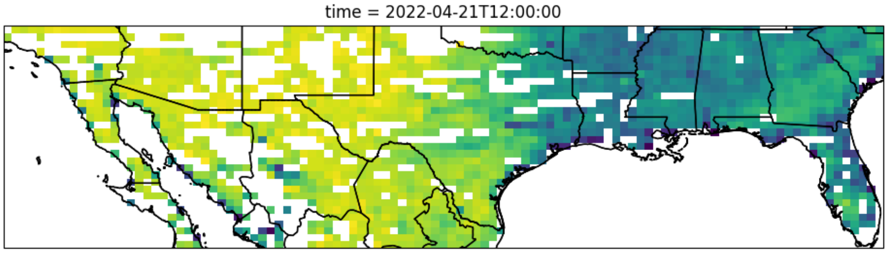
```python
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature

fig,ax = plt.subplots(ncols=1,nrows=1, figsize=(11.69,4), subplot_kw={'projection': ccrs.PlateCarree()})
ds_cygnss_.SM_daily.isel(time=110).plot(ax=ax, cmap='viridis')
ax.coastlines()
plt.savefig('fig1_gaps.png', dpi=500)
```



```
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from matplotlib.patches import Rectangle

# Create the plot with Cartopy projection
fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(11.69, 4), subplot_kw={'projection': ccrs.PlateCarree()})
sm_plot = ds_cygnss_.SM_daily.isel(time=110).plot(ax=ax, cmap='viridis_r', cbar_kwargs={'shrink': 0.6, 'label': 'Soil Moistu

# Add coastlines
ax.coastlines()

# Add title
ax.set_title('Soil Moisture at 21 April 2022 12:00:00')

# Add black box (lat = slice(25, 35), lon = slice(-120, -90))
lon_min, lon_max = -120, -80
lat_min, lat_max = 25, 35

# Create a Rectangle for the box (black border, no fill)
box = Rectangle((lon_min, lat_min), lon_max - lon_min, lat_max - lat_min,
                linewidth=2, edgecolor='black', facecolor='none', transform=ccrs.PlateCarree())

# Add the rectangle to the plot
ax.add_patch(box)

# Modify colorbar and add text
cbar = sm_plot.colorbar
cbar.set_label('Soil Moisture (m³/m³)', fontsize=10)
cbar.ax.text(1.05, 0.5, 'High Moisture\n\n\n\n\n\n\n\n\n\n\n\nLow Moisture', transform=cbar.ax.transAxes, va='center')

# Save the figure
plt.savefig('fig1_gaps_1.png', dpi=500)
```

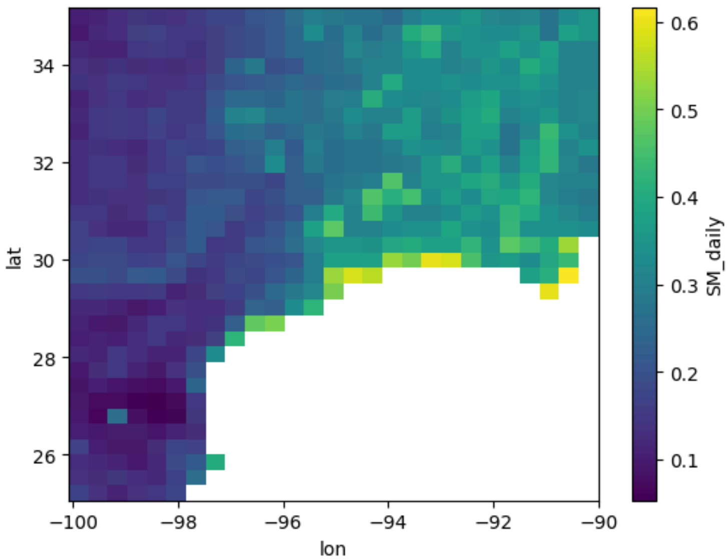



```
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature

fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(11.69, 8), subplot_kw={'projection': ccrs.PlateCarree()})
sm_plot = ds_cygnss_.SM_daily.isel(time=110).sel(lat=slice(25,35)).sel(lon=slice(-120,-80)).plot(ax=ax, cmap='viridis_r', ac

ax.coastlines()
ax.add_feature(cfeature.STATES.with_scale('10m'))  # Add state borders
ax.add_feature(cfeature.BORDERS, linestyle=':')
# ax.set_title('Soil Moisture at 21 April 2022 12:00:00')

# # Adding text on the colorbar (near its label)
# cbar = sm_plot.colorbar
# cbar.set_label('Soil Moisture (m³/m³)', fontsize=10)
# cbar.ax.text(1.05, 0.5, 'High Moisture\n\n\n\n\n\n\n\n\n\n\n\nLow Moisture', transform=cbar.ax.transAxes, va='center')

plt.savefig('fig1_gaps_2.png', dpi=500)
```




```
1 ds_cygnss_.sel(lat=slice(25,35)).sel(lon=slice(-100,-90)).SM_daily.mean(dim='time').plot()
```

```
<matplotlib.collections.QuadMesh at 0x78d95a2d7510>
```

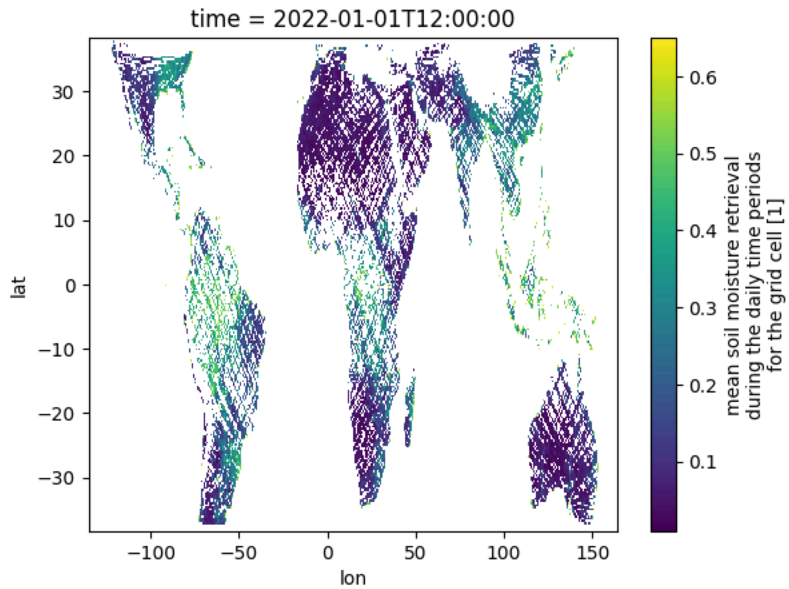



```
1 ds_cygnss_tx = ds_cygnss_.sel(lat=slice(25,35)).sel(lon=slice(-100,-90)).SM_daily
2 ds_cygnss_tx = ds_cygnss_.SM_daily
3 ds_cygnss_tx.shape
```

```
(360, 252, 802)
```

```
1 ds_cygnss_tx.isel(time=0).plot()
```

```
<matplotlib.collections.QuadMesh at 0x78d95afaca50>
```

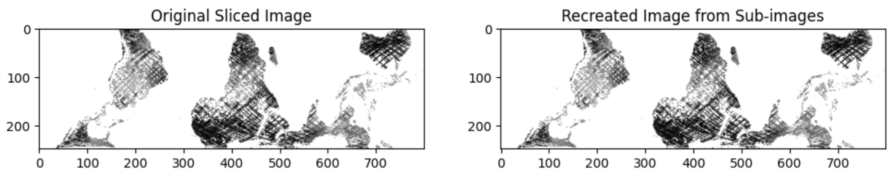


Make sub images

```python
import numpy as np
import xarray as xr
import matplotlib.pyplot as plt
from skimage.util import view_as_blocks

def slice_data_for_blocks(data, block_size=(64, 64)):
    """
    Slice the data to ensure dimensions are divisible by the block size.

    Parameters:
    data (xarray.DataArray): The original data array.
    block_size (tuple): The size of the blocks (sub-images).

    Returns:
    xarray.DataArray: The sliced data array.
    tuple: The shape of the cropped data array.
    """
    lat_size, lon_size = data.shape[-2], data.shape[-1]
    lat_new = (lat_size // block_size[0]) * block_size[0]
    lon_new = (lon_size // block_size[1]) * block_size[1]
    data_sliced = data.isel(lat=slice(0, lat_new), lon=slice(0, lon_new))
    return data_sliced, (lat_new, lon_new)

def generate_sub_images(data, block_size=(64, 64)):
    """
    Generate sub-images of the specified block size from the original image.

    Parameters:
    data (numpy array): The original image array of shape (H, W).
    block_size (tuple): The size of the blocks (sub-images) to generate.

    Returns:
    numpy array: An array of sub-images.
    """
    H, W = data.shape
    blocks = view_as_blocks(data, block_shape=block_size)
    sub_images = blocks.reshape(-1, *block_size)
    return sub_images

def recreate_image_from_sub_images(sub_images, original_shape, block_size=(64, 64)):
    """
    Recreate the original image from sub-images.

    Parameters:
    sub_images (numpy array): An array of sub-images.
    original_shape (tuple): The shape of the cropped image (H, W).
    block_size (tuple): The size of the blocks (sub-images) used.

    Returns:
    numpy array: The recreated original image.
    """
    H, W = original_shape
    h_blocks = H // block_size[0]
    w_blocks = W // block_size[1]
    sub_images = sub_images.reshape(h_blocks, w_blocks, *block_size)
    recreated_image = np.block([[sub_images[i, j] for j in range(w_blocks)] for i in range(h_blocks)])
    return recreated_image

# Ensure data dimensions are divisible by block size
data_sliced, cropped_shape = slice_data_for_blocks(ds_cygnss_tx, block_size=(8, 8))

# Generate sub-images
sub_images = generate_sub_images(data_sliced[0,:,:].values, block_size=(8, 8))



# Recreate the original image
recreated_image = recreate_image_from_sub_images(sub_images, cropped_shape, block_size=(8, 8))

print(f"Original slice shape: {ds_cygnss_tx.shape}")
print(f"Sliced data shape: {data_sliced.shape}")
print(f"Sub-images shape: {sub_images.shape}")
print(f"Recreated image shape: {recreated_image.shape}")

# Visualization
fig, ax = plt.subplots(1, 2, figsize=(12, 6))

```

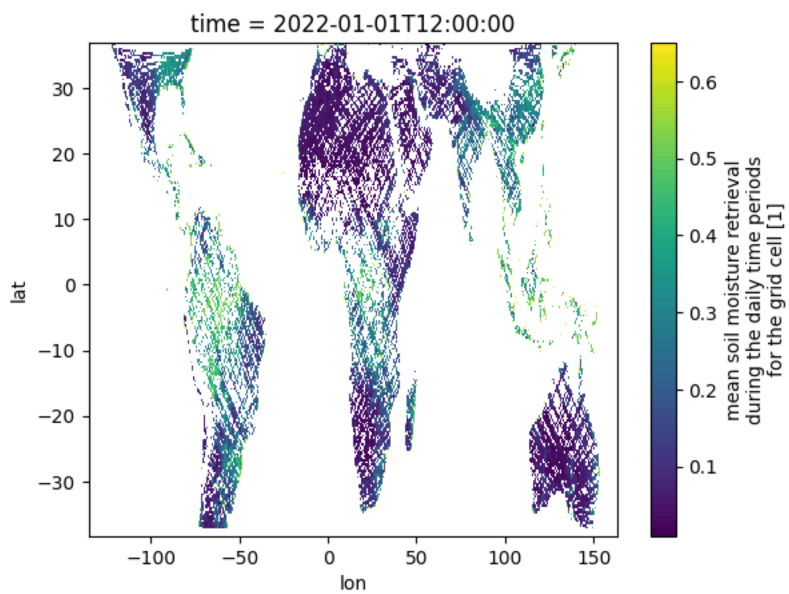
```
ax[0].imshow(data_sliced[0,:,:], cmap='gray')
ax[0].set_title('Original Sliced Image')

ax[1].imshow(recreated_image, cmap='gray')
ax[1].set_title('Recreated Image from Sub-images')

plt.show()
```

```
Original slice shape: (360, 252, 802)
Sliced data shape: (360, 248, 800)
Sub-images shape: (3100, 8, 8)
Recreated image shape: (248, 800)
```




```
data_sliced.isel(time=0).plot()
```

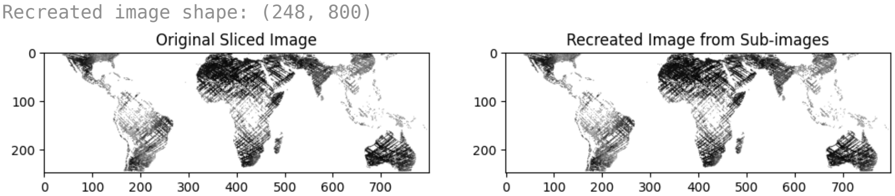
```
<matplotlib.collections.QuadMesh at 0x78d979f3c910>
```



```
import numpy as np
import xarray as xr
import matplotlib.pyplot as plt
from skimage.util import view_as_blocks

def slice_data_for_blocks(data, block_size=(64, 64)):
    """
    Slice the data to ensure dimensions are divisible by the block size.

    Parameters:
    data (xarray.DataArray): The original data array.
    block_size (tuple): The size of the blocks (sub-images).

    Returns:
    xarray.DataArray: The sliced data array.
    tuple: The shape of the cropped data array.
    """
    lat_size, lon_size = data.shape[-2], data.shape[-1]
    lat_new = (lat_size // block_size[0]) * block_size[0]
    lon_new = (lon_size // block_size[1]) * block_size[1]
    data_sliced = data.isel(lat=slice(0, lat_new), lon=slice(0, lon_new))
    return data_sliced, (lat_new, lon_new)

def generate_sub_images(data, block_size=(64, 64)):
    """
    Generate sub-images of the specified block size from the original image.

    Parameters:
    data (numpy array): The original image array of shape (H, W).
    block_size (tuple): The size of the blocks (sub-images) to generate.

    Returns:
    numpy array: An array of sub-images.
    """
    H, W = data.shape
    blocks = view_as_blocks(data, block_shape=block_size)
    sub_images = blocks.reshape(-1, *block_size)
    return sub_images

def recreate_image_from_sub_images(sub_images, original_shape, block_size=(64, 64)):
    """
    Recreate the original image from sub-images.

    Parameters:
    sub_images (numpy array): An array of sub-images.
    original_shape (tuple): The shape of the cropped image (H, W).
    block_size (tuple): The size of the blocks (sub-images) used.

    Returns:
    numpy array: The recreated original image.
    """
    H, W = original_shape
    h_blocks = H // block_size[0]
    w_blocks = W // block_size[1]
    sub_images = sub_images.reshape(h_blocks, w_blocks, *block_size)
    recreated_image = np.block([[sub_images[i, j] for j in range(w_blocks)] for i in range(h_blocks)])
    return recreated_image

def visualize_sub_images(sub_images, block_size=(64, 64), ncols=8):
    """
    Visualize the sub-images.

    Parameters:
    sub_images (numpy array): An array of sub-images.
    block_size (tuple): The size of the blocks (sub-images).
    ncols (int): The number of columns for visualization.
    """
    nrows = int(np.ceil(len(sub_images) / ncols))
    fig, axes = plt.subplots(nrows, ncols, figsize=(11, 8))
    for ax, sub_image in zip(axes.flat, sub_images):
        ax.imshow(sub_image, cmap='gray')
        ax.axis('off')
    for ax in axes.flat[len(sub_images):]:
        ax.axis('off')
    plt.savefig('sub_images_before_training.png', dpi=500)

data_array = ds_cygnss_tx #ds_combined.isel(time=times_to_isel).Band1
```

```python
# Select a specific time slice
data_slice = ds_cygnss_tx #data_array.sel(time=data_array.time[0])

# Ensure data dimensions are divisible by block size
data_sliced, cropped_shape = slice_data_for_blocks(data_slice, block_size=(8, 8))

# Convert to numpy and flip along vertical axis to correct orientation
data_sliced_np = np.flipud(data_sliced.values)

# Generate sub-images
sub_images = generate_sub_images(data_sliced_np[0,:,:], block_size=(8, 8))

# Recreate the original image
recreated_image = recreate_image_from_sub_images(sub_images, cropped_shape, block_size=(8, 8))

# Flip the recreated image back to the original orientation
recreated_image = np.flipud(recreated_image)

print(f"Original slice shape: {data_slice.shape}")
print(f"Sliced data shape: {data_sliced.shape}")
print(f"Sub-images shape: {sub_images.shape}")
print(f"Recreated image shape: {recreated_image.shape}")

# Visualization
fig, ax = plt.subplots(1, 2, figsize=(12, 6))

ax[0].imshow(np.flipud(data_sliced_np[0,:,:]), cmap='gray')
ax[0].set_title('Original Sliced Image')

ax[1].imshow(recreated_image, cmap='gray')
ax[1].set_title('Recreated Image from Sub-images')

plt.show()

# # Visualize sub-images
# visualize_sub_images(sub_images, block_size=(64, 64), ncols=8)
```

```
Original slice shape: (360, 252, 802)
Sliced data shape: (360, 248, 800)
Sub-images shape: (3100, 8, 8)
Recreated image shape: (248, 800)
```

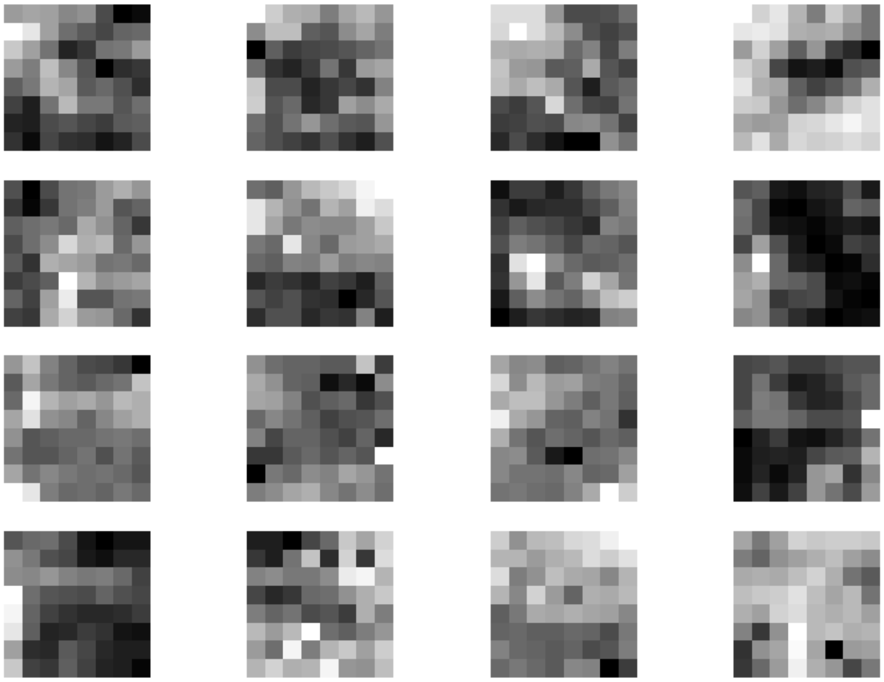


```
def remove_nan_sub_images(sub_images):
    """
    Remove sub-images that contain NaN values.

    Parameters:
    sub_images (numpy array): An array of sub-images.

    Returns:
    numpy array: Filtered array of sub-images without NaN values.
    """
    mask = ~np.isnan(sub_images).any(axis=(1, 2))
    return sub_images[mask]

# Assuming ds_combined is already loaded and times_to_isel is defined
data_array = ds_cygnss_tx # ds_combined.isel(time=times_to_isel).Band1

# Process each time slice
all_sub_images = []
for time in data_array.time:
    data_slice = data_array.sel(time=time)

    # Ensure data dimensions are divisible by block size
    data_sliced, cropped_shape = slice_data_for_blocks(data_slice, block_size=(8, 8))

    # Convert to numpy and flip along vertical axis to correct orientation
    data_sliced_np = np.flipud(data_sliced.values)

    # Generate sub-images
    sub_images = generate_sub_images(data_sliced_np, block_size=(8, 8))

    # Remove sub-images with NaN values
    sub_images = remove_nan_sub_images(sub_images)

    all_sub_images.append(sub_images)

# Combine all sub-images into a single array for visualization if needed
all_sub_images = np.concatenate(all_sub_images, axis=0)
```

```
print(all_sub_images.shape, np.min(all_sub_images), np.max(all_sub_images))
```

```
(1178, 8, 8) 0.010150299 0.60524344
```

```
# Visualize sub-images
visualize_sub_images(all_sub_images[:16], block_size=(4, 4), ncols=4)
```



```
!mv sub_images_before_training.png fig2_sub_images_without_nans.png
```

Gap Filling Starts

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
import numpy as np
import matplotlib.pyplot as plt
```

```python
# Define the U-Net architecture
class UNet(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(UNet, self).__init__()

        def conv_block(in_channels, out_channels):
            block = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
                nn.ReLU(inplace=True),
                nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1),
                nn.ReLU(inplace=True)
            )
            return block

        self.encoder1 = conv_block(in_channels, 64)
        self.encoder2 = conv_block(64, 128)
        self.encoder3 = conv_block(128, 256)
        self.encoder4 = conv_block(256, 512)

        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        self.middle = conv_block(512, 1024)

        self.upconv4 = nn.ConvTranspose2d(1024, 512, kernel_size=2, stride=2)
        self.decoder4 = conv_block(1024, 512)
        self.upconv3 = nn.ConvTranspose2d(512, 256, kernel_size=2, stride=2)
        self.decoder3 = conv_block(512, 256)
        self.upconv2 = nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2)
        self.decoder2 = conv_block(256, 128)
        self.upconv1 = nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2)
        self.decoder1 = conv_block(128, 64)

        self.final_conv = nn.Conv2d(64, out_channels, kernel_size=1)

    def forward(self, x):
        enc1 = self.encoder1(x)
        enc2 = self.encoder2(self.pool(enc1))
        enc3 = self.encoder3(self.pool(enc2))
        enc4 = self.encoder4(self.pool(enc3))

        middle = self.middle(self.pool(enc4))

        dec4 = self.upconv4(middle)
        dec4 = torch.cat((dec4, enc4), dim=1)
        dec4 = self.decoder4(dec4)

        dec3 = self.upconv3(dec4)
        dec3 = torch.cat((dec3, enc3), dim=1)
        dec3 = self.decoder3(dec3)

        dec2 = self.upconv2(dec3)
        dec2 = torch.cat((dec2, enc2), dim=1)
        dec2 = self.decoder2(dec2)

        dec1 = self.upconv1(dec2)
        dec1 = torch.cat((dec1, enc1), dim=1)
        dec1 = self.decoder1(dec1)

        out = self.final_conv(dec1)
        return out

class FSRCNN(nn.Module):
    def __init__(self, d=56, s=12, m=4, upscale_factor=1):
        super(FSRCNN, self).__init__()
        self.first_part = nn.Sequential(
            nn.Conv2d(2, d, kernel_size=5, padding=5//2),
            nn.PReLU(d)
        )

        self.mid_parts = [nn.Sequential(
            nn.Conv2d(d, s, kernel_size=1),
            nn.PReLU(s)
        )]

        for _ in range(m - 1):
            self.mid_parts.append(nn.Sequential(
                nn.Conv2d(s, s, kernel_size=3, padding=3//2),
```

```python
                nn.PReLU(s)
            ))

        self.mid_parts = nn.Sequential(*self.mid_parts)

        self.last_part = nn.Sequential(
            nn.Conv2d(s, d, kernel_size=1),
            nn.PReLU(d),
            nn.ConvTranspose2d(d, 1, kernel_size=9, stride=upscale_factor, padding=9//2, output_padding=upscale_factor-1)
        )

    def forward(self, x):
        x = self.first_part(x)
        x = self.mid_parts(x)
        x = self.last_part(x)
        return x
```

```python
# Define the custom dataset
class CYGNSSDataset(Dataset):
    def __init__(self, incomplete_data, complete_data, transform=None):
        self.incomplete_data = incomplete_data
        self.complete_data = complete_data
        self.transform = transform

    def __len__(self):
        return len(self.incomplete_data)

    def __getitem__(self, idx):
        incomplete = self.incomplete_data[idx]
        complete = self.complete_data[idx]

        if self.transform:
            incomplete = self.transform(incomplete)
            complete = self.transform(complete)

        return incomplete, complete

# Masking function to simulate missing values
def mask_data(data, missing_percentage):
    masked_data = data.copy()
    mask = np.random.rand(*data.shape) < missing_percentage
    masked_data[mask] = np.nan  # Use NaN for missing values
    return masked_data

# Create a mask channel where 1 indicates valid data and 0 indicates missing data
def create_mask(data):
    mask = ~np.isnan(data)
    return mask.astype(np.float32)

# Impute NaN values with the mean of the non-NaN values
def impute_data(data):
    nan_mask = np.isnan(data)
    mean_value = np.nanmean(data)
    data[nan_mask] = mean_value
    return data
```

```python
all_sub_images[:,np.newaxis,:,:].shape
```

```
(1178, 1, 8, 8)
```

```python
print(all_sub_images.shape, np.min(all_sub_images), np.max(all_sub_images))
```

```
(1178, 8, 8) 0.010150299 0.60524344
```

```python
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from torch.utils.data import DataLoader, Dataset, Subset
import joblib

complete_data = all_sub_images[:,np.newaxis,:,:]

# Save the scaler parameters to disk
scaler_params = {'min': np.min(all_sub_images), 'max': np.max(all_sub_images)}
# scaler_filename = "/content/drive/MyDrive/EarthEngineExports/"+"scaler_params.joblib"
```

```
# scaler_filename = '/content/drive/MyDrive/EarthEngineExports/' + scaler_params.joblib
# joblib.dump(scaler_params, scaler_filename)

# # Load the scaler parameters from disk (for demonstration purposes)
# loaded_params = joblib.load(scaler_filename)
# loaded_min = loaded_params['min']
# loaded_max = loaded_params['max']

# Manually apply min-max scaling using the loaded parameters
scaled_complete_data = 2 * (complete_data - scaler_params['min']) / (scaler_params['max'] - scaler_params['min'])
scaled_complete_data = scaled_complete_data
```

```
scaled_complete_data.shape
```

```
(1178, 1, 8, 8)
```

```
incomplete_data = mask_data(scaled_complete_data, missing_percentage=0.3)  # 30% missing values
print(incomplete_data.shape)
# Create mask and impute data
masks = np.array([create_mask(img) for img in incomplete_data])
incomplete_data = np.array([impute_data(img) for img in incomplete_data])
print(incomplete_data.shape)
# Combine the imputed data and the mask into a two-channel input
combined_data = np.concatenate((incomplete_data, masks), axis=1)
print(combined_data.shape)
```

```
(1178, 1, 8, 8)
(1178, 1, 8, 8)
(1178, 2, 8, 8)
```

```
complete_data
```

```
       [[[0.06679883, 0.06567191, 0.06627198, ..., 0.03026018,
          0.03147613, 0.03873805],
         [0.0655243 , 0.07459313, 0.06598178, ..., 0.03198734,
          0.02765077, 0.0315891 ],
         [0.05444514, 0.05373578, 0.05574022, ..., 0.04372385,
          0.02903239, 0.04269865],
         ...,
         [0.03038205, 0.02665055, 0.03291544, ..., 0.02927097,
          0.02753259, 0.04062437],
         [0.02623009, 0.02824694, 0.03188973, ..., 0.04692882,
          0.04040937, 0.02816401],
         [0.0223181 , 0.02967042, 0.02329663, ..., 0.01129153,
          0.04740182, 0.03971875]]],


       ...,
```

```
[[[0.06388564, 0.06162073, 0.06293219, ..., 0.02396213,
   0.0334433 , 0.05403574],
  [0.06561529, 0.06274135, 0.06710986, ..., 0.06439031,
   0.05253886, 0.04894675],
  [0.05933033, 0.06275615, 0.06064529, ..., 0.04996726,
   0.05940418, 0.05089023],
  ...,
  [0.04379449, 0.05184331, 0.03935004, ..., 0.04130233,
   0.04944286, 0.05897599],
  [0.05291521, 0.04786249, 0.05135668, ..., 0.0475975 ,
```

```
# Create the dataset
dataset = CYGNSSDataset(combined_data, complete_data)

# Define the split ratios
train_ratio = 0.7
val_ratio = 0.15
test_ratio = 0.15

# Split the dataset indices
train_size = int(train_ratio * len(dataset))
val_size = int(val_ratio * len(dataset))
test_size = len(dataset) - train_size - val_size

train_indices, temp_indices = train_test_split(range(len(dataset)), train_size=train_size, shuffle=True)
val_indices, test_indices = train_test_split(temp_indices, test_size=test_size, shuffle=True)

# Create subset datasets
train_dataset = Subset(dataset, train_indices)
val_dataset = Subset(dataset, val_indices)
test_dataset = Subset(dataset, test_indices)

# Create DataLoaders
batch_size = 4
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

print(f"Train dataset size: {len(train_dataset)}")
print(f"Validation dataset size: {len(val_dataset)}")
print(f"Test dataset size: {len(test_dataset)}")
```

```
Train dataset size: 824
Validation dataset size: 176
Test dataset size: 178
```

```
# Model, loss function, and optimizer
#model = UNet(in_channels=2, out_channels=1).cuda()  # Update to handle 2-channel input
model = FSRCNN().cuda()
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
print(dataset.__getitem__(0)[0].shape, combined_data.astype(np.float32).shape)
```

```
(2, 8, 8) (1178, 2, 8, 8)
```

```
!cp /content/drive/MyDrive/EarthEngineExports/best_model_cygnss_gap_fill.pth .
```

```
import torch
import torch.nn as nn
import torch.optim as optim

# Define DataLoader for validation data
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)

# Training the model
num_epochs = 1000
best_val_loss = float('inf')
best_model_path = "/content/drive/MyDrive/EarthEngineExports/" + "best_model_cygnss_gap_fill.pth"
train = False
if train:
    for epoch in range(num_epochs):
        model.train()
        for i, (incomplete, complete) in enumerate(train_loader):
            incomplete = incomplete.float().cuda()
            complete = complete.float().cuda()

            # Forward pass
            outputs = model(incomplete)
            loss = criterion(outputs, complete)

            # Backward pass and optimization
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            if (i + 1) % 10 == 0:
                print(f'Epoch [{epoch+1}/{num_epochs}], Step [{i+1}/{len(train_loader)}], Loss: {loss.item():.4f}')

        # Evaluate on validation data
        model.eval()
        val_loss = 0.0
        with torch.no_grad():
            for incomplete, complete in val_loader:
                incomplete = incomplete.float().cuda()
                complete = complete.float().cuda()

                outputs = model(incomplete)
                loss = criterion(outputs, complete)
                val_loss += loss.item()

        val_loss /= len(val_loader)
        print(f'Epoch [{epoch+1}/{num_epochs}], Validation Loss: {val_loss:.4f}')

        # Save the model if validation loss has decreased
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            torch.save(model.state_dict(), best_model_path)
            print(f'Saved best model with validation loss: {best_val_loss:.4f}')

# Load the best model for further use or evaluation
model.load_state_dict(torch.load(best_model_path))
print('Loaded best model for further use or evaluation')
```

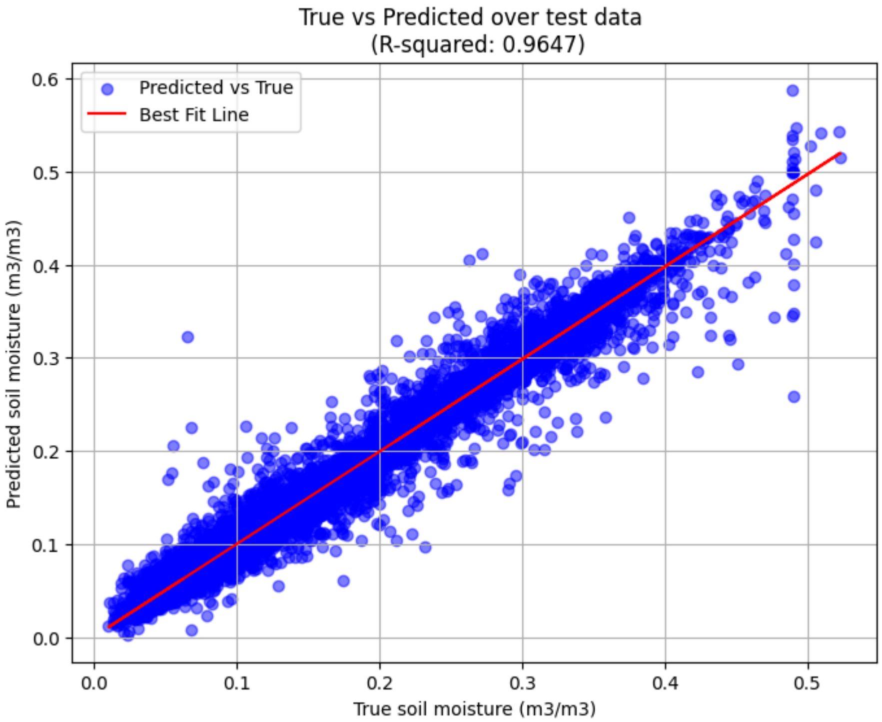
```
Loaded best model for further use or evaluation
```

```python
import numpy as np
from sklearn.metrics import r2_score
import torch

# Evaluation loop with R-squared calculation
def evaluate_model_with_r2(model, dataloader, criterion):
    model.eval()
    eval_loss = 0.0
    all_targets = []
    all_outputs = []

    with torch.no_grad():
        for inputs, targets in dataloader:
            # Convert inputs and targets to tensors and move to GPU
            inputs = torch.tensor(inputs).float().cuda()
            targets = torch.tensor(targets).float().cuda()

            # Ensure inputs have the correct number of channels
            if inputs.shape[1] == 1:  # If input has 1 channel, expand it to 2 channels
                inputs = inputs.repeat(1, 2, 1, 1)

            outputs = model(inputs)
            loss = criterion(outputs, targets)

            eval_loss += loss.item() * inputs.size(0)

            all_targets.append(targets.cpu().numpy())
            all_outputs.append(outputs.cpu().numpy())

    eval_loss /= len(dataloader.dataset)
    print(f'Evaluation Loss: {eval_loss:.4f}')

    # Concatenate all batches
    all_targets = np.concatenate(all_targets, axis=0).reshape(-1)
    all_outputs = np.concatenate(all_outputs, axis=0).reshape(-1)

    r2 = r2_score(all_targets, all_outputs)
    print(f'R-squared: {r2:.4f}')
    return eval_loss, r2

# Assuming test_loader is already defined
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# Instantiate and run the evaluation
evaluate_model_with_r2(model, test_loader, criterion)
```

```
Evaluation Loss: 0.0003
R-squared: 0.9694
/tmp/ipython-input-184-3314331158.py:15: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor
  inputs = torch.tensor(inputs).float().cuda()
/tmp/ipython-input-184-3314331158.py:16: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor
  targets = torch.tensor(targets).float().cuda()
(0.0003112466790676745, 0.9693548083305359)
```

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import r2_score
import torch

# Evaluation loop with R-squared calculation and plotting
def evaluate_model_with_r2_and_plots(model, dataloader, criterion):
    model.eval()
    eval_loss = 0.0
    all_targets = []
    all_outputs = []

    with torch.no_grad():
        for inputs, targets in dataloader:
            # Convert inputs and targets to tensors and move to GPU
            inputs = torch.tensor(inputs).float().cuda()
            targets = torch.tensor(targets).float().cuda()

            # Ensure inputs have the correct number of channels
            if inputs.shape[1] == 1:  # If input has 1 channel, expand it to 2 channels
                inputs = inputs.repeat(1, 2, 1, 1)

            outputs = model(inputs)
            loss = criterion(outputs, targets)

            eval_loss += loss.item() * inputs.size(0)

            all_targets.append(targets.cpu().numpy())
            all_outputs.append(outputs.cpu().numpy())

    eval_loss /= len(dataloader.dataset)
    print(f'Evaluation Loss: {eval_loss:.4f}')

    # Concatenate all batches
    all_targets = np.concatenate(all_targets, axis=0).reshape(-1)
    all_outputs = np.concatenate(all_outputs, axis=0).reshape(-1)

    # Calculate R-squared
    r2 = r2_score(all_targets, all_outputs)
    print(f'R-squared: {r2:.4f}')

    # Plot scatter plot with best fit line
    plot_results(all_targets, all_outputs)

    return eval_loss, r2

# Function to create scatter plot with best fit line
def plot_results(true_values, predicted_values):
    plt.figure(figsize=(8, 6))

    # Scatter plot
    plt.scatter(true_values, predicted_values, color='blue', label='Predicted vs True', alpha=0.5)

    # Best fit line
    best_fit_line = np.poly1d(np.polyfit(true_values, predicted_values, 1))
    plt.plot(true_values, best_fit_line(true_values), color='red', label='Best Fit Line')

    # Plot labels
    plt.xlabel('True soil moisture (m3/m3)')
    plt.ylabel('Predicted soil moisture (m3/m3)')
    plt.title('True vs Predicted over test data \n (R-squared: 0.9647)')
    plt.legend()

    # Show plot
    plt.grid(True)
    plt.savefig('best_fit.png', dpi=500)

# Assuming test_loader is already defined
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# Instantiate and run the evaluation with plots
evaluate_model_with_r2_and_plots(model, test_loader, criterion)

```

```
/tmp/ipython-input-185-977835508.py:16: UserWarning: To copy construct from a te
  inputs = torch.tensor(inputs).float().cuda()
/tmp/ipython-input-185-977835508.py:17: UserWarning: To copy construct from a te
  targets = torch.tensor(targets).float().cuda()
Evaluation Loss: 0.0003
R-squared: 0.9694
(0.00031124667694211385, 0.9693548083305359)
```




```
!mv best_fit.png fig3_best_fit.png
```

Generate Predictions

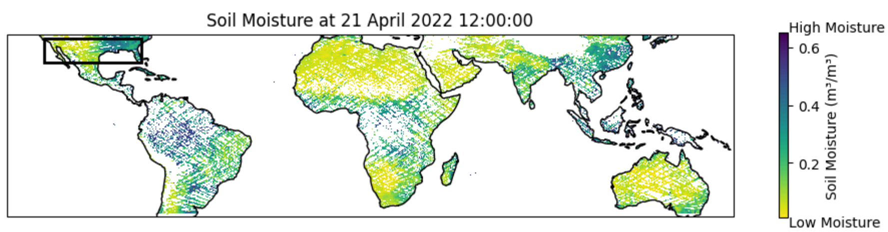
```
incomplete_data = ds_cygnss_tx.isel(time=110).data
incomplete_data.shape
```

```
(252, 802)
```

```
incomplete_data = ds_cygnss_tx.isel(time=110).data[:248, :800]
print(incomplete_data[np.newaxis].shape, np.sum(np.isnan(incomplete_data[np.newaxis])))

scaled_incomplete_data = 2 * (incomplete_data - scaler_params['min']) / (scaler_params['max'] - scaler_params['min'])
scaled_incomplete_data = scaled_incomplete_data#.compute()

print(scaled_incomplete_data.shape)
# Create mask and impute data
masks = np.array([create_mask(img) for img in scaled_incomplete_data])
incomplete_data = np.array([impute_data(img) for img in scaled_incomplete_data])
print(masks.shape)
# Combine the imputed data and the mask into a two-channel input
combined_data = np.concatenate((incomplete_data[np.newaxis], masks[np.newaxis]), axis=0)
print(combined_data.shape)

gap_filled = model(torch.tensor(combined_data[np.newaxis]).float().cuda()).cpu().detach().numpy()[0,0]
gap_filled.shape
```

```
(1, 248, 800) 166519
(248, 800)
(248, 800)
(2, 248, 800)
/tmp/ipython-input-172-1112879747.py:36: RuntimeWarning: Mean of empty slice
  mean_value = np.nanmean(data)
(248, 800)
```

```
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from matplotlib.patches import Rectangle

# Create the plot with Cartopy projection
fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(11.69, 4), subplot_kw={'projection': ccrs.PlateCarree()})
sm_plot = ds_cygnss_.SM_daily.isel(time=110).isel(lat=slice(0,248)).isel(lon=slice(0,800)).plot(ax=ax, cmap='viridis_r', cba

# Add coastlines
ax.coastlines()

# Add title
ax.set_title('Soil Moisture at 21 April 2022 12:00:00')

# Add black box (lat = slice(25, 35), lon = slice(-120, -90))
lon_min, lon_max = -120, -80
lat_min, lat_max = 25, 35

# Create a Rectangle for the box (black border, no fill)
box = Rectangle((lon_min, lat_min), lon_max - lon_min, lat_max - lat_min,
                linewidth=2, edgecolor='black', facecolor='none', transform=ccrs.PlateCarree())

# Add the rectangle to the plot
ax.add_patch(box)

# Modify colorbar and add text
cbar = sm_plot.colorbar
cbar.set_label('Soil Moisture (m³/m³)', fontsize=10)
cbar.ax.text(1.05, 0.5, 'High Moisture\n\n\n\n\n\n\n\n\n\n\nLow Moisture', transform=cbar.ax.transAxes, va='center')

# Save the figure
plt.savefig('fig1_gaps_1.png', dpi=500)
```




```
ds_cygnss_gap_fill = ds_cygnss_tx.isel(time=110).isel(lat=slice(0,248)).isel(lon=slice(0,800))
ds_cygnss_gap_fill['SM_daily_gap_filled'] = (('lat', 'lon'), gap_filled)
ds_cygnss_gap_fill
```

xarray.DataArray 'SM_daily' (**lat**: 248, **lon**: 800)

```
array([[nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       ...,
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan]], dtype=float32)
```

▾ Coordinates:

time	()	datetime64[ns]	2022-04-21T12:00:00
lat	(lat)	float32	-38.14 -37.79 ... 36.38 36.73
lon	(lon)	float32	-135.0 -134.6 ... 163.0 163.4
SM_daily_gap_f...	(lat, lon)	float32	nan nan nan ... 0.2705 0.2405

▸ Indexes: (2)

▾ Attributes:

comment :	units represent soil moisture content as a fractional volume (cm3 cm-3)
long_name :	mean soil moisture retrieval during the daily time periods for the grid cell
units :	1
coverage_conte...	modelResult

```python
mask = ds_cygnss_.SM_daily.mean(dim='time').isel(lat=slice(0,248)).isel(lon=slice(0,800)).values
mask = mask/mask
mask
```

```
array([[nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       ...,
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan]], dtype=float32)
```

```python
ds_cygnss_gap_fill_mask = ds_cygnss_gap_fill.SM_daily_gap_filled*mask
ds_cygnss_gap_fill_mask = ds_cygnss_gap_fill_mask*(scaler_params['max'] - scaler_params['min'])/2 + scaler_params['min']
ds_cygnss_gap_fill_mask = ((ds_cygnss_gap_fill_mask - ds_cygnss_gap_fill_mask.mean())/ds_cygnss_gap_fill_mask.std()) *ds_cyg
ds_cygnss_gap_fill_mask
```

xarray.DataArray (**lat**: 248, **lon**: 800)

```
array([[nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       ...,
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan]])
```

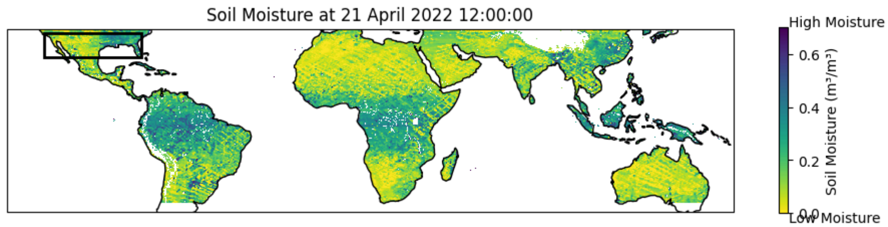
▼ Coordinates:

time	()	datetime64[ns]	2022-04-21T12:00:00
lat	(lat)	float32	-38.14 -37.79 ... 36.38 36.73
lon	(lon)	float32	-135.0 -134.6 ... 163.0 163.4
SM_daily_gap_f...	(lat, lon)	float32	nan nan nan ... 0.2705 0.2405

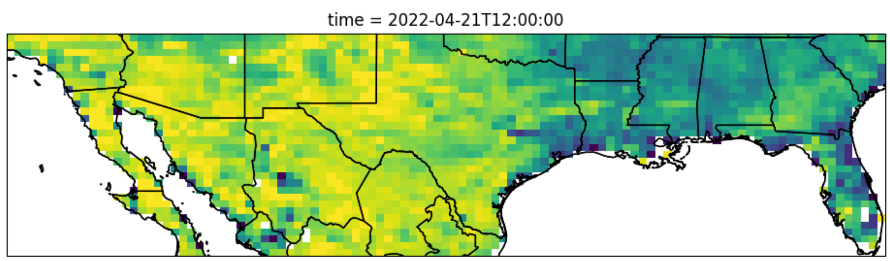
▸ Indexes: (2)

▸ Attributes: (0)

```python
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from matplotlib.patches import Rectangle

# Create the plot with Cartopy projection
fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(11.69, 4), subplot_kw={'projection': ccrs.PlateCarree()})
sm_plot = ds_cygnss_gap_fill_mask.plot(ax=ax, cmap='viridis_r', cbar_kwargs={'shrink': 0.6, 'label': 'Soil Moisture (m³/m³)'}

# Add coastlines
ax.coastlines()

# Add title
ax.set_title('Soil Moisture at 21 April 2022 12:00:00')

# Add black box (lat = slice(25, 35), lon = slice(-120, -90))
lon_min, lon_max = -120, -80
lat_min, lat_max = 25, 35

# Create a Rectangle for the box (black border, no fill)
box = Rectangle((lon_min, lat_min), lon_max - lon_min, lat_max - lat_min,
                linewidth=2, edgecolor='black', facecolor='none', transform=ccrs.PlateCarree())

# Add the rectangle to the plot
ax.add_patch(box)

# Modify colorbar and add text
cbar = sm_plot.colorbar
cbar.set_label('Soil Moisture (m³/m³)', fontsize=10)
cbar.ax.text(1.05, 0.5, 'High Moisture\n\n\n\n\n\n\n\n\n\n\nLow Moisture', transform=cbar.ax.transAxes, va='center')

# Save the figure
plt.savefig('fig4_gap_filled_1.png', dpi=500)
```




```python
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature

fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(11.69, 8), subplot_kw={'projection': ccrs.PlateCarree()})
sm_plot = ds_cygnss_gap_fill_mask.sel(lat=slice(25,35)).sel(lon=slice(-120,-80)).plot(ax=ax, cmap='viridis_r', add_colorbar=F

ax.coastlines()
ax.add_feature(cfeature.STATES.with_scale('10m'))  # Add state borders
ax.add_feature(cfeature.BORDERS, linestyle=':')

plt.savefig('fig4_gap_filled_2.png', dpi=500)
```



```
import numpy as np
import xarray as xr
import torch
from tqdm import tqdm

# Helper functions (reuse from before)
def create_mask(data):
    return (~np.isnan(data)).astype(np.float32)

def impute_data(data):
    data = data.copy()
    nan_mask = np.isnan(data)
    data[nan_mask] = np.nanmean(data)
    return data

# Subset CYGNSS data (adjust region if needed)
lat_slice = slice(0, 248)
lon_slice = slice(0, 800)
ds_crop = ds_cygnss_tx.isel(lat=lat_slice, lon=lon_slice)

# Initialize empty list for results
gap_filled_all = []

# Loop over time dimension
for t in tqdm(range(ds_crop.shape[0]), desc="Gap-filling over time"):
    # Extract soil moisture at time t
    incomplete_data = ds_crop.isel(time=t).values

    # Skip if fully missing
    if np.isnan(incomplete_data).all():
        gap_filled_all.append(np.full_like(incomplete_data, np.nan))
        continue

    # Scale
    scaled = 2 * (incomplete_data - scaler_params['min']) / (scaler_params['max'] - scaler_params['min'])

    # Mask and impute
    mask = create_mask(scaled)
    scaled = impute_data(scaled)

    # Stack as 2-channel input
    combined = np.stack([scaled, mask], axis=0)  # Shape: [2, H, W]

    # Model prediction
    with torch.no_grad():
        input_tensor = torch.tensor(combined[np.newaxis]).float().cuda()  # Shape: [1, 2, H, W]
        output_tensor = model(input_tensor).cpu().numpy()[0, 0]  # Shape: [H, W]

    # Append output
    gap_filled_all.append(output_tensor)

# Convert to numpy array
gap_filled_all = np.array(gap_filled_all)  # Shape: [time, lat, lon]

# Create xarray DataArray
ds_gap_filled = xr.DataArray(
    gap_filled_all,
    dims=['time', 'lat', 'lon'],
    coords={
        'time': ds_crop.time.values,
        'lat': ds_crop.lat.values,
        'lon': ds_crop.lon.values
    },
    name="SM_daily_gap_filled"
)

# Optional: save to NetCDF
ds_gap_filled.to_netcdf("cygnss_gap_filled_all.nc")
print("Gap-filled data saved to 'cygnss_gap_filled_all.nc'")
```

```
Gap-filling over time: 100%|██████████| 360/360 [00:02<00:00, 121.85it/s]
Gap-filled data saved to 'cygnss_gap_filled_all.nc'
```

```
import xarray as xr
import numpy as np

# Subset and crop the data (same region as deep learning gap-fill)
lat_slice = slice(0, 248)
lon_slice = slice(0, 800)
ds_interp_input = ds_cygnss_tx.isel(lat=lat_slice, lon=lon_slice)

# Apply linear interpolation along the time dimension
ds_linear_interp = ds_interp_input.interpolate_na(
    dim="time",
    method="linear",
    fill_value="extrapolate"  # Optional: extrapolates at edges
)

# Rename for consistency
ds_linear_interp.name = "SM_daily_linear_interp"

# Save the result
ds_linear_interp.to_netcdf("cygnss_gap_filled_linear.nc")
print("Linear interpolation gap-filled data saved to 'cygnss_gap_filled_linear.nc'")
```

```
Linear interpolation gap-filled data saved to 'cygnss_gap_filled_linear.nc'
```

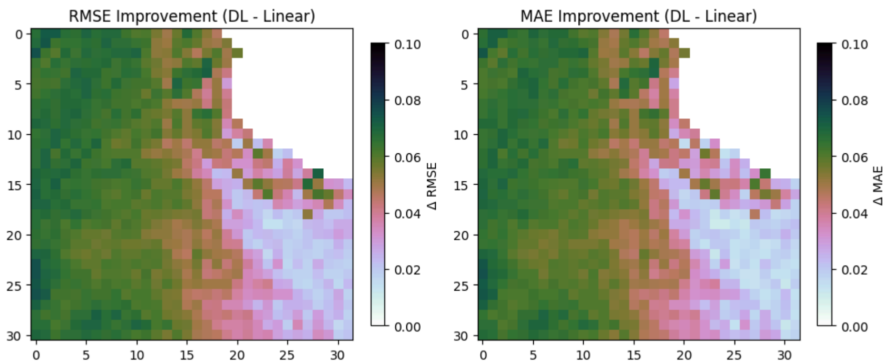
```
!pip install scikit-image
```

```
Requirement already satisfied: scikit-image in /usr/local/lib/python3.11/dist-packages (0.25.2)
Requirement already satisfied: numpy>=1.24 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (2.0.2)
Requirement already satisfied: scipy>=1.11.4 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (1.15.3)
Requirement already satisfied: networkx>=3.0 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (3.5)
Requirement already satisfied: pillow>=10.1 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (11.2.1)
Requirement already satisfied: imageio!=2.35.0,>=2.33 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (2.37.0
Requirement already satisfied: tifffile>=2022.8.12 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (2025.6.11
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (24.2)
Requirement already satisfied: lazy-loader>=0.4 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (0.4)
```

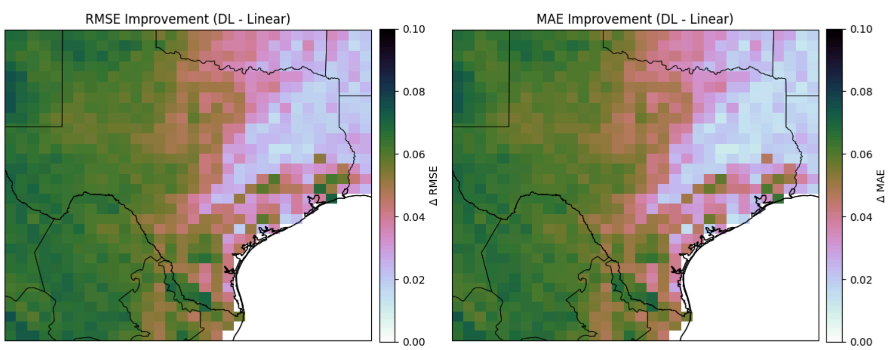
```
Start coding or generate with AI.
```

```
import numpy as np
import xarray as xr
from sklearn.metrics import mean_squared_error, mean_absolute_error
from skimage.metrics import structural_similarity as ssim
import matplotlib.pyplot as plt

# Define the region of interest (Texas in this case)
lat_slice = slice(25, 35)
lon_slice = slice(-105, -93)

# Extract observed (original), DL gap-filled, and Linear interpolated datasets
obs = ds_cygnss_tx.sel(lat=lat_slice, lon=lon_slice)
dl = ds_gap_filled.sel(lat=lat_slice, lon=lon_slice)
li = ds_linear_interp.sel(lat=lat_slice, lon=lon_slice)

# Align all three over time
obs, dl, li = xr.align(obs, dl, li)

# Initialize metric maps
lat_size = obs.sizes['lat']
lon_size = obs.sizes['lon']

rmse_dl_map = np.full((lat_size, lon_size), np.nan)
mae_dl_map = np.full((lat_size, lon_size), np.nan)
ssim_dl_map = np.full((lat_size, lon_size), np.nan)

rmse_li_map = np.full((lat_size, lon_size), np.nan)
mae_li_map = np.full((lat_size, lon_size), np.nan)
ssim_li_map = np.full((lat_size, lon_size), np.nan)

# Loop over each grid point
for i in range(lat_size):
    for j in range(lon_size):
        obs_series = obs[:, i, j].values
        dl_series = dl[:, i, j].values
        li_series = li[:, i, j].values

        # Only evaluate at times where observations exist
        mask = ~np.isnan(obs_series)
        if np.sum(mask) < 10:
            continue

        obs_valid = obs_series[mask]
        dl_valid = dl_series[mask]
        li_valid = li_series[mask]

        # Skip if DL or LI has NaNs (could happen at edge or masked regions)
        if np.isnan(dl_valid).any() or np.isnan(li_valid).any():
            continue

        # DL metrics
        rmse_dl_map[i, j] = np.sqrt(mean_squared_error(obs_valid, dl_valid))
        mae_dl_map[i, j] = mean_absolute_error(obs_valid, dl_valid)
        try:
            ssim_dl_map[i, j] = ssim(obs_valid.reshape(-1, 1), dl_valid.reshape(-1, 1), data_range=obs_valid.max() - obs_val
        except:
            pass

        # LI metrics
        rmse_li_map[i, j] = np.sqrt(mean_squared_error(obs_valid, li_valid))
        mae_li_map[i, j] = mean_absolute_error(obs_valid, li_valid)
        try:
            ssim_li_map[i, j] = ssim(obs_valid.reshape(-1, 1), li_valid.reshape(-1, 1), data_range=obs_valid.max() - obs_val
        except:
            pass
```

```
import matplotlib.pyplot as plt

fig, axs = plt.subplots(1, 2, figsize=(10, 5))

# RMSE improvement (DL - Linear)
diff_rmse = (rmse_li_map - rmse_dl_map) * -1  # Positive = DL is better
im1 = axs[0].imshow(diff_rmse, cmap='cubehelix_r', vmin=0.0, vmax=0.1)
axs[0].set_title('RMSE Improvement (DL - Linear)')
plt.colorbar(im1, ax=axs[0], fraction=0.04, label='Δ RMSE')

# MAE improvement (DL - Linear)
diff_mae = (mae_li_map - mae_dl_map) * -1
im2 = axs[1].imshow(diff_mae, cmap='cubehelix_r', vmin=0.0, vmax=0.1)
axs[1].set_title('MAE Improvement (DL - Linear)')
plt.colorbar(im2, ax=axs[1], fraction=0.04, label='Δ MAE')

plt.tight_layout()
plt.savefig("fig7_dl_vs_linear_vs_observed_metrics.png", dpi=500)
```



```
lat = obs.lat.values
lon = obs.lon.values

# Now construct the xarray DataArrays using lat/lon
da_rmse = xr.DataArray(diff_rmse, coords=[('lat', lat), ('lon', lon)])
da_mae = xr.DataArray(diff_mae, coords=[('lat', lat), ('lon', lon)])

import xarray as xr
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature

# Get lat/lon from obs grid
lat = obs.lat.values
lon = obs.lon.values

# Compute improvement
diff_rmse = (rmse_li_map - rmse_dl_map) * -1
diff_mae = (mae_li_map - mae_dl_map) * -1

# Wrap into xarray
da_rmse = xr.DataArray(diff_rmse, coords=[('lat', lat), ('lon', lon)])
da_mae = xr.DataArray(diff_mae, coords=[('lat', lat), ('lon', lon)])

# Plot
fig, axs = plt.subplots(1, 2, figsize=(12, 6), subplot_kw={'projection': ccrs.PlateCarree()})

im1 = da_rmse.plot(ax=axs[0], cmap='cubehelix_r', vmin=0.0, vmax=0.1, add_colorbar=False)
axs[0].set_title('RMSE Improvement (DL - Linear)')
axs[0].coastlines()
axs[0].add_feature(cfeature.STATES, linewidth=0.5)
axs[0].add_feature(cfeature.BORDERS, linestyle=':')

im2 = da_mae.plot(ax=axs[1], cmap='cubehelix_r', vmin=0.0, vmax=0.1, add_colorbar=False)
axs[1].set_title('MAE Improvement (DL - Linear)')
axs[1].coastlines()
axs[1].add_feature(cfeature.STATES, linewidth=0.5)
axs[1].add_feature(cfeature.BORDERS, linestyle=':')

fig.colorbar(im1, ax=axs[0], orientation='vertical', fraction=0.04, pad=0.02, label='Δ RMSE')
fig.colorbar(im2, ax=axs[1], orientation='vertical', fraction=0.04, pad=0.02, label='Δ MAE')

plt.tight_layout()
plt.savefig("fig7_dl_vs_linear_vs_observed_metrics.png", dpi=500, bbox_inches='tight')
plt.show()
```



```
# Central Texas drought grid (near Travis County)
lat_drought, lon_drought = 30.3, -97.8

# DFW flood grid (near Dallas)
lat_flood, lon_flood = 32.8, -96.8
```

```
# Dates to visualize
event_dates = {
    "Drought_July15": "2022-07-15",
    "Flood_Aug22": "2022-08-22"
}
```

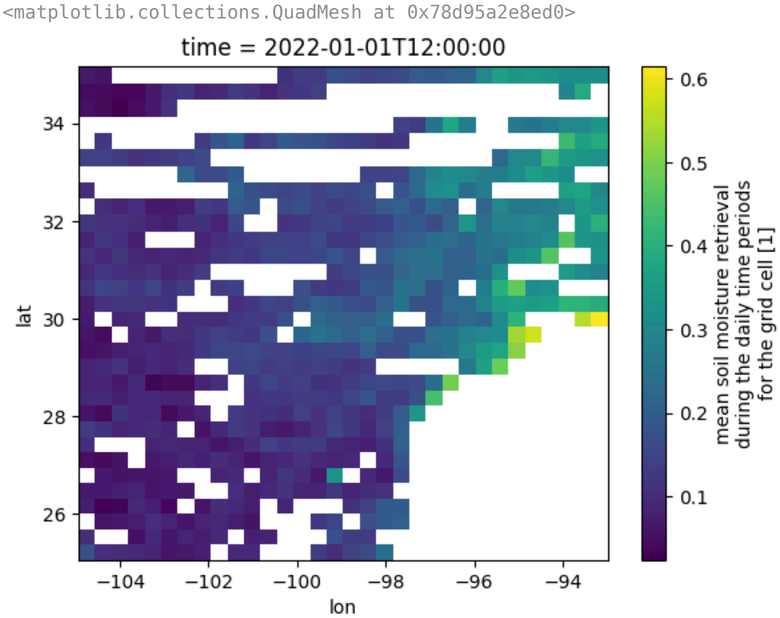
```
!pip install cartopy
```

```
Requirement already satisfied: cartopy in /usr/local/lib/python3.11/dist-packages (0.24.1)
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/python3.11/dist-packages (from cartopy) (2.0.2)
Requirement already satisfied: matplotlib>=3.6 in /usr/local/lib/python3.11/dist-packages (from cartopy) (3.10.0)
Requirement already satisfied: shapely>=1.8 in /usr/local/lib/python3.11/dist-packages (from cartopy) (2.1.1)
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.11/dist-packages (from cartopy) (24.2)
Requirement already satisfied: pyshp>=2.3 in /usr/local/lib/python3.11/dist-packages (from cartopy) (2.3.1)
Requirement already satisfied: pyproj>=3.3.1 in /usr/local/lib/python3.11/dist-packages (from cartopy) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy) (
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy) (0.12
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy) (
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartop
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from pyproj>=3.3.1->cartopy) (2025.6.15)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib>=3
```

```
import cartopy.crs as ccrs
```

```
obs.isel(time=0).plot()
```

```
<matplotlib.collections.QuadMesh at 0x78d95a2e8ed0>
```




```
mask = obs.mean(dim='time').values
mask = mask/mask
mask
```

```
        [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1., nan, nan, nan],
        [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
          1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
```

```
1 ds_cygnss_gap_fill_final = xr.open_dataset('cygnss_gap_filled_all.nc')
```

```
1 obs
```

xarray.DataArray 'SM_daily' (**time**: 360, **lat**: 31, **lon**: 32)

```
array([[[0.17282172, 0.10677953, 0.09998017, ...,        nan,
                nan,        nan],
        [0.09763184,        nan,        nan, ...,        nan,
                nan,        nan],
        [       nan, 0.05690542, 0.06908599, ...,        nan,
                nan,        nan],
        ...,
        [0.0495064 , 0.04074128, 0.02374071, ...,        nan,
                nan,        nan],
        [0.04680497, 0.03953968, 0.03390655, ..., 0.294771  ,
         0.38873288,        nan],
        [0.06561837, 0.05833703,        nan, ..., 0.31498757,
         0.31073517, 0.31847593]],

       [[       nan, 0.1187866 , 0.07842803, ...,        nan,
                nan,        nan],
        [0.08966997, 0.05844343, 0.05773357, ...,        nan,
                nan,        nan],
        [0.10128325, 0.06714807,        nan, ...,        nan,
                nan,        nan],
...
        [       nan,        nan,        nan, ..., 0.30785176,
         0.36155513, 0.40475845],
        [       nan,        nan,        nan, ...,        nan,
                nan,        nan],
        [       nan,        nan,        nan, ..., 0.28684017,
         0.33965173, 0.35877475]],

       [[0.17062363, 0.08910783, 0.08385277, ...,        nan,
                nan,        nan],
        [       nan, 0.07862744, 0.0920056 , ...,        nan,
                nan,        nan],
        [0.08699363, 0.05127537,        nan, ...,        nan,
                nan,        nan],
        ...,
        [       nan,        nan,        nan, ..., 0.24178925,
         0.38094625, 0.40376106],
        [       nan,        nan, 0.08295673, ...,        nan,
                nan,        nan],
        [       nan,        nan,        nan, ..., 0.27071264,
         0.31280124, 0.3380334 ]]], dtype=float32)
```

▼ Coordinates:

time	(time)	datetime64[ns]	2022-01-01T12:00:00 ... 2022-12-...
lat	(lat)	float32	25.22 25.53 25.84 ... 34.65 34.99
lon	(lon)	float32	-104.8 -104.4 ... -93.55 -93.17

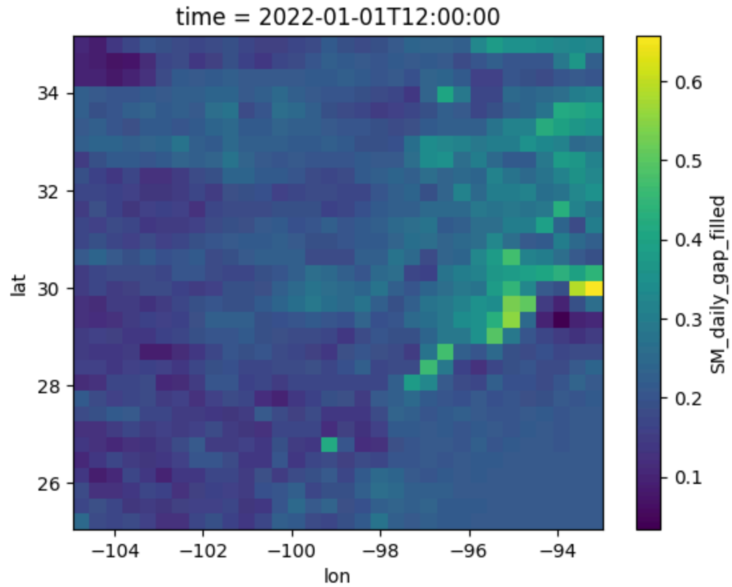
▸ Indexes: (3)

▼ Attributes:

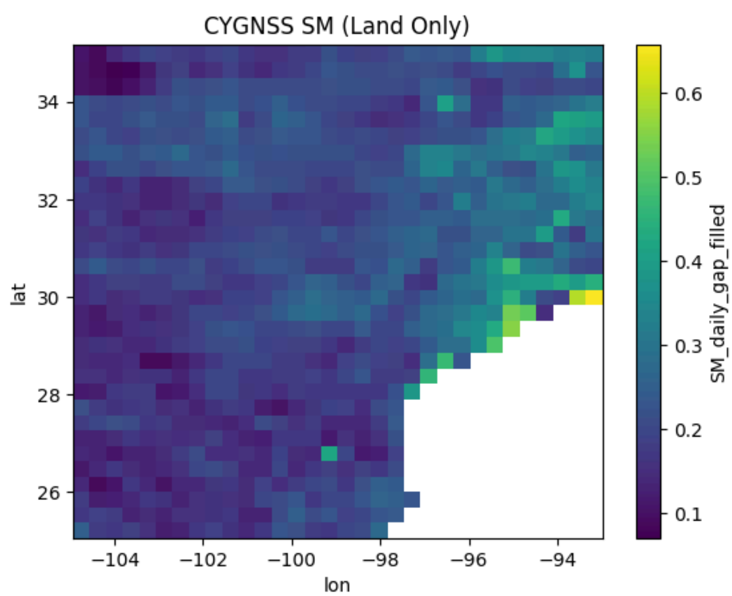
comment :	units represent soil moisture content as a fractional volume (cm3 cm-3)
long_name :	mean soil moisture retrieval during the daily time periods for the grid cell
units :	1
coverage_conte…	modelResult

```
1 ds_cygnss_gap_fill_final.SM_daily_gap_filled.sel(lat =slice(obs.lat.values[0], obs.lat.values[-1])).sel(lon =slice(obs.lon.\
```

```
<matplotlib.collections.QuadMesh at 0x78d95af713d0>
```




```
import xarray as xr
import numpy as np
import matplotlib.pyplot as plt

# Extract the first time step
sm_data = ds_cygnss_gap_fill_final.SM_daily_gap_filled.sel(
    lat=slice(obs.lat.values[0], obs.lat.values[-1]),
    lon=slice(obs.lon.values[0], obs.lon.values[-1])
).isel(time=0)

# Assume you have land_mask as 2D (lat, lon), same shape as sm_data
# Apply the mask: set ocean (mask==0) to NaN
masked_sm = sm_data.where(mask == 1)

# Plot
masked_sm.plot()
plt.title("CYGNSS SM (Land Only)")
plt.show()
```

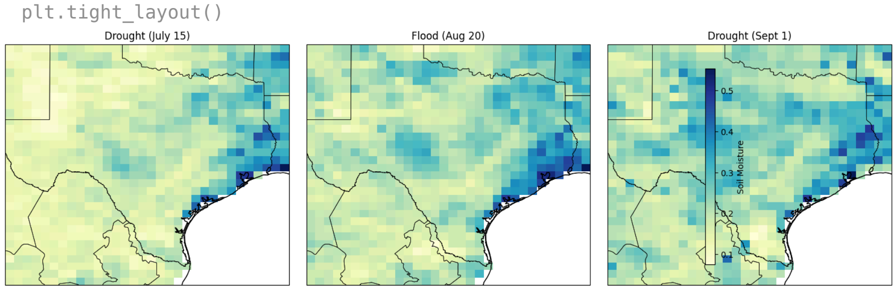


```
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import matplotlib.pyplot as plt

# Define event dates (can be extended)
event_dict = {
    "Drought_July15": "2022-07-15",
    "Flood_Aug20": "2022-08-20",
    "Drought_Sept01": "2022-09-01"
}

# Directory to save figures
output_dir = "./cygnss_event_maps/"
import os
os.makedirs(output_dir, exist_ok=True)

for event_name, date_str in event_dict.items():
    # Get data
    sm_data = ds_cygnss_gap_fill_final.SM_daily_gap_filled.sel(
        lat=slice(obs.lat.values[0], obs.lat.values[-1]),
        lon=slice(obs.lon.values[0], obs.lon.values[-1]),
        time=date_str
    )

    # Apply land mask
    masked_sm = sm_data.where(mask == 1)

    # Plot with Cartopy
    fig = plt.figure(figsize=(10, 6))
    ax = plt.axes(projection=ccrs.PlateCarree())
    im = masked_sm.plot(ax=ax, transform=ccrs.PlateCarree(), cmap="YlGnBu", cbar_kwargs={"label": "Soil Moisture"})

    # Add features
    ax.coastlines()
    ax.add_feature(cfeature.BORDERS, linestyle=':')
    ax.add_feature(cfeature.STATES, linewidth=0.5)
    ax.set_title(f"CYGNSS SM during {event_name.replace('_', ' ')}", fontsize=14)

    # Save
    filepath = os.path.join(output_dir, f"{event_name}.png")
    plt.savefig(filepath, dpi=300, bbox_inches='tight')
    plt.close()

    print(f"Saved: {filepath}")
```

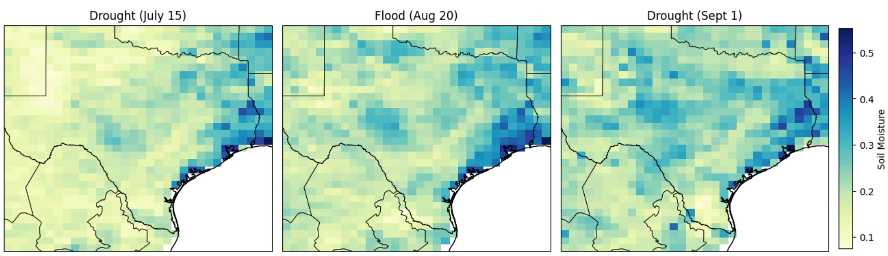
```
Saved: ./cygnss_event_maps/Drought_July15.png
Saved: ./cygnss_event_maps/Flood_Aug20.png
Saved: ./cygnss_event_maps/Drought_Sept01.png
```

```
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import matplotlib.pyplot as plt

# Define event dates and titles
event_dict = {
    "2022-07-15": "Drought (July 15)",
    "2022-08-20": "Flood (Aug 20)",
    "2022-09-01": "Drought (Sept 1)"
}

# Create figure
n_events = len(event_dict)
fig, axs = plt.subplots(1, n_events, figsize=(5 * n_events, 6),
                        subplot_kw={'projection': ccrs.PlateCarree()})

if n_events == 1:
    axs = [axs]  # ensure it's iterable if only one subplot

# Loop over events
for ax, (date_str, title) in zip(axs, event_dict.items()):
    sm_data = ds_cygnss_gap_fill_final.SM_daily_gap_filled.sel(
        lat=slice(obs.lat.values[0], obs.lat.values[-1]),
        lon=slice(obs.lon.values[0], obs.lon.values[-1]),
        time=date_str
    )
    masked_sm = sm_data.where(mask == 1)


    im = masked_sm.plot(ax=ax, transform=ccrs.PlateCarree(), cmap="YlGnBu",
                        add_colorbar=False)  # suppress colorbar per plot
    ax.coastlines()
    ax.add_feature(cfeature.BORDERS, linestyle=':')
    ax.add_feature(cfeature.STATES, linewidth=0.5)
    ax.set_title(f"{title}", fontsize=12)

# Add a shared colorbar
cbar = fig.colorbar(im, ax=axs, orientation='vertical', shrink=0.7, label="Soil Moisture")

# Save and show
plt.tight_layout()
plt.savefig("CYGNSS_SM_Events_2022.png", dpi=300)
plt.show()
```

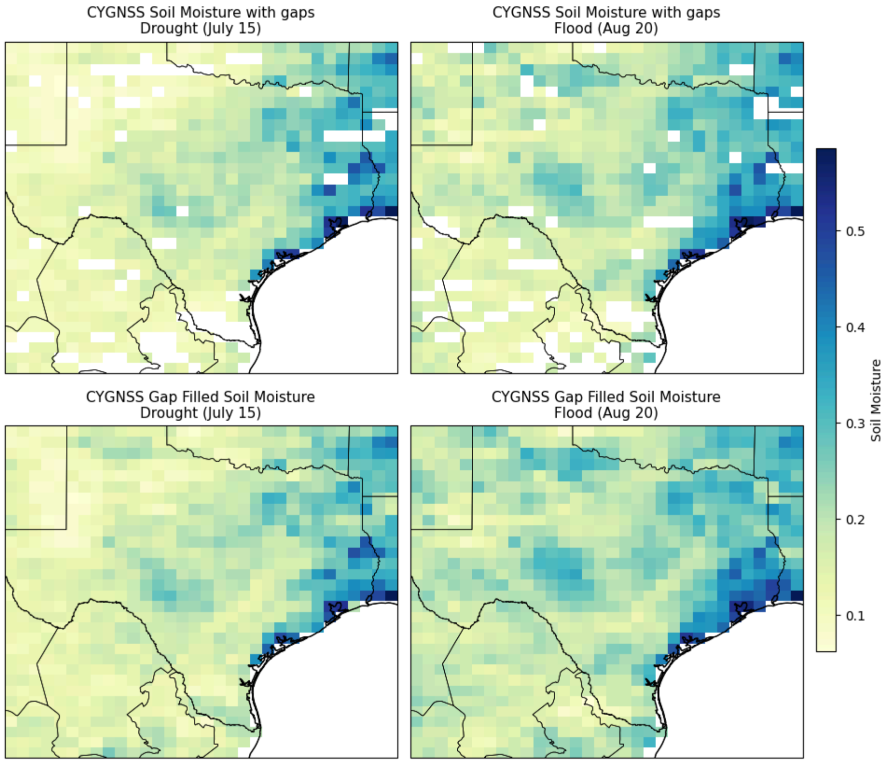
```
/tmp/ipython-input-212-476118328.py:41: UserWarning: This figure includes Axes t
  plt.tight_layout()
```



```python
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import matplotlib.pyplot as plt
from matplotlib import gridspec

# Define event dates and titles
event_dict = {
    "2022-07-15": "Drought (July 15)",
    "2022-08-20": "Flood (Aug 20)",
    "2022-09-01": "Drought (Sept 1)"
}

# Set up figure with space on right for colorbar
n_events = len(event_dict)
fig = plt.figure(figsize=(5 * n_events + 1, 6))
gs = gridspec.GridSpec(1, n_events + 1, width_ratios=[1]*n_events + [0.05], wspace=0.05)

axs = []
for i in range(n_events):
    ax = fig.add_subplot(gs[0, i], projection=ccrs.PlateCarree())
    axs.append(ax)

# Plot each event
for ax, (date_str, title) in zip(axs, event_dict.items()):
    sm_data = ds_cygnss_gap_fill_final.SM_daily_gap_filled.sel(
        lat=slice(obs.lat.values[0], obs.lat.values[-1]),
        lon=slice(obs.lon.values[0], obs.lon.values[-1]),
        time=date_str
    )
    masked_sm = sm_data.where(mask == 1)

    im = masked_sm.plot(ax=ax, transform=ccrs.PlateCarree(), cmap="YlGnBu",
                        add_colorbar=False)
    ax.coastlines()
    ax.add_feature(cfeature.BORDERS, linestyle=':')
    ax.add_feature(cfeature.STATES, linewidth=0.5)
    ax.set_title(title, fontsize=12)

# Add vertical colorbar with shorter height
cax = fig.add_subplot(gs[0, -1])
pos = cax.get_position()
# Shrink to 70% of original height and center
cax.set_position([pos.x0, pos.y0 + pos.height * 0.15, pos.width, pos.height * 0.7])

cbar = fig.colorbar(im, cax=cax, orientation='vertical')
cbar.set_label("Soil Moisture")

# Save and show
plt.savefig("CYGNSS_SM_Events_2022.png", dpi=300, bbox_inches='tight')
plt.show()
```

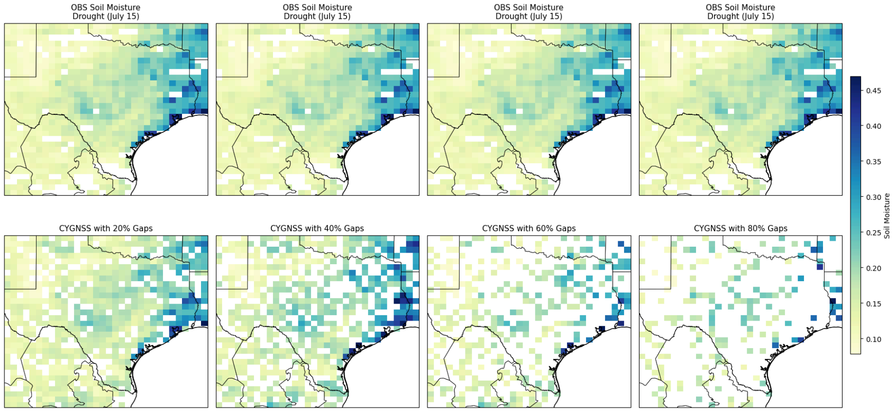


```
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import matplotlib.pyplot as plt
from matplotlib import gridspec

# Define event dates and titles
event_dict = {
    "2022-07-15": "Drought (July 15)",
    "2022-08-20": "Flood (Aug 20)"
}

n_events = len(event_dict)

# Create 2-row grid: Row 1 = OBS, Row 2 = CYGNSS
fig = plt.figure(figsize=(5 * n_events + 1.5, 10))
gs = gridspec.GridSpec(2, n_events + 1, width_ratios=[1]*n_events + [0.05], hspace=0.15, wspace=0.05)

axs_obs = []
axs_dl = []

# Plot observations (top row)
for i, (date_str, title) in enumerate(event_dict.items()):
    ax = fig.add_subplot(gs[0, i], projection=ccrs.PlateCarree())
    axs_obs.append(ax)

    obs_data = obs.sel(time=date_str)
    im_obs = obs_data.plot(ax=ax, transform=ccrs.PlateCarree(), cmap="YlGnBu",
                           add_colorbar=False)

    ax.coastlines()
    ax.add_feature(cfeature.BORDERS, linestyle=':')
    ax.add_feature(cfeature.STATES, linewidth=0.5)
    ax.set_title(f"CYGNSS Soil Moisture with gaps\n{title}", fontsize=11)

# Plot CYGNSS (bottom row)
for i, (date_str, title) in enumerate(event_dict.items()):
    ax = fig.add_subplot(gs[1, i], projection=ccrs.PlateCarree())
    axs_dl.append(ax)

    sm_data = ds_cygnss_gap_fill_final.SM_daily_gap_filled.sel(
        lat=slice(obs.lat.values[0], obs.lat.values[-1]),
        lon=slice(obs.lon.values[0], obs.lon.values[-1]),
        time=date_str
    )
    masked_sm = sm_data.where(mask == 1)

    im_dl = masked_sm.plot(ax=ax, transform=ccrs.PlateCarree(), cmap="YlGnBu",
                           add_colorbar=False)

    ax.coastlines()
    ax.add_feature(cfeature.BORDERS, linestyle=':')
    ax.add_feature(cfeature.STATES, linewidth=0.5)
    ax.set_title(f"CYGNSS Gap Filled Soil Moisture\n{title}", fontsize=11)

# Colorbar outside
cax = fig.add_subplot(gs[:, -1])
pos = cax.get_position()
cax.set_position([pos.x0, pos.y0 + pos.height * 0.15, pos.width, pos.height * 0.7])
cbar = fig.colorbar(im_dl, cax=cax, orientation='vertical')
cbar.set_label("Soil Moisture")

# Save and show
plt.savefig("Obs_vs_CYGNSS_SM_2022.png", dpi=300, bbox_inches='tight')
plt.show()
```

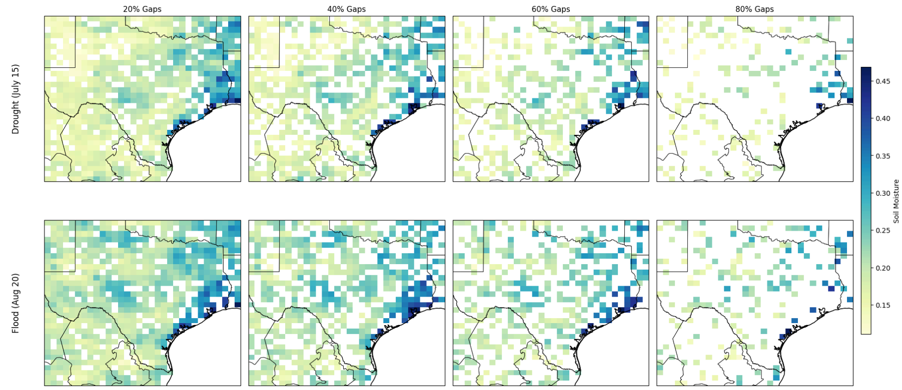


More validation

```
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import matplotlib.pyplot as plt
from matplotlib import gridspec
import numpy as np

# Required for plotting in Colab
%matplotlib inline

# Define a single event
event_date = "2022-07-15"
event_title = "Drought (July 15)"
gap_levels = [0.2, 0.4, 0.6, 0.8]

# Create 2-row grid: Row 1 = Original OBS, Row 2 = CYGNSS with different gaps
fig = plt.figure(figsize=(5 * len(gap_levels) + 1.5, 10))
gs = gridspec.GridSpec(2, len(gap_levels) + 1, width_ratios=[1]*len(gap_levels) + [0.05], hspace=0.15, wspace=0.05)

axs_obs = []
axs_dl = []

# Row 1: Plot same OBS data for all columns
for i, gap in enumerate(gap_levels):
    ax = fig.add_subplot(gs[0, i], projection=ccrs.PlateCarree())
    axs_obs.append(ax)

    obs_data = obs.sel(time=event_date)
    im_obs = obs_data.plot(ax=ax, transform=ccrs.PlateCarree(), cmap="YlGnBu", add_colorbar=False)

    ax.coastlines()
    ax.add_feature(cfeature.BORDERS, linestyle=':')
    ax.add_feature(cfeature.STATES, linewidth=0.5)
    ax.set_title(f"OBS Soil Moisture\n{event_title}", fontsize=11)

# Row 2: CYGNSS with artificial gaps at different levels
for i, gap in enumerate(gap_levels):
    ax = fig.add_subplot(gs[1, i], projection=ccrs.PlateCarree())
    axs_dl.append(ax)

    sm_data = ds_cygnss_gap_fill_final.SM_daily_gap_filled.sel(
        lat=slice(obs.lat.values[0], obs.lat.values[-1]),
        lon=slice(obs.lon.values[0], obs.lon.values[-1]),
        time=event_date
    )

    masked_sm = sm_data.where(mask == 1)

    # Create artificial gaps
    rng = np.random.default_rng(seed=42 + i)  # different seed per gap
    gap_mask = rng.uniform(size=masked_sm.shape) > gap  # keep (1 - gap)
    masked_sm = masked_sm.where(gap_mask)

    im_dl = masked_sm.plot(ax=ax, transform=ccrs.PlateCarree(), cmap="YlGnBu", add_colorbar=False)

    ax.coastlines()
    ax.add_feature(cfeature.BORDERS, linestyle=':')
    ax.add_feature(cfeature.STATES, linewidth=0.5)
    ax.set_title(f"CYGNSS with {int(gap*100)}% Gaps", fontsize=11)

# Shared colorbar
cax = fig.add_subplot(gs[:, -1])
pos = cax.get_position()
cax.set_position([pos.x0, pos.y0 + pos.height * 0.15, pos.width, pos.height * 0.7])
cbar = fig.colorbar(im_dl, cax=cax, orientation='vertical')
cbar.set_label("Soil Moisture")

# Save and show
plt.savefig("CYGNSS_gap_levels_comparison.png", dpi=300, bbox_inches='tight')
plt.show()
```


OBS Soil Moisture
Drought (July 15)
OBS Soil Moisture
Drought (July 15)
OBS Soil Moisture
Drought (July 15)
OBS Soil Moisture
Drought (July 15)
CYGNSS with 20% Gaps
CYGNSS with 40% Gaps
CYGNSS with 60% Gaps
CYGNSS with 80% Gaps
Soil Moisture
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10

```python
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import matplotlib.pyplot as plt
from matplotlib import gridspec
import numpy as np

# Required for plotting in Colab
%matplotlib inline

# Define events and gap levels
event_dict = {
    "2022-07-15": "Drought (July 15)",
    "2022-08-20": "Flood (Aug 20)"
}
gap_levels = [0.2, 0.4, 0.6, 0.8]

# Create figure grid: 2 rows (events), 4 cols (gap levels), plus 1 for colorbar
fig = plt.figure(figsize=(5 * len(gap_levels) + 1.5, 10))
gs = gridspec.GridSpec(len(event_dict), len(gap_levels) + 1,
                       width_ratios=[1]*len(gap_levels) + [0.05],
                       hspace=0.15, wspace=0.05)

# Iterate over each event
for row_idx, (date_str, title) in enumerate(event_dict.items()):
    for col_idx, gap in enumerate(gap_levels):
        ax = fig.add_subplot(gs[row_idx, col_idx], projection=ccrs.PlateCarree())

        sm_data = ds_cygnss_gap_fill_final.SM_daily_gap_filled.sel(
            lat=slice(obs.lat.values[0], obs.lat.values[-1]),
            lon=slice(obs.lon.values[0], obs.lon.values[-1]),
            time=date_str
        )

        masked_sm = sm_data.where(mask == 1)

        # Create artificial gaps
        rng = np.random.default_rng(seed=100 * row_idx + col_idx)  # unique seed
        gap_mask = rng.uniform(size=masked_sm.shape) > gap  # keep (1 - gap)
        masked_sm = masked_sm.where(gap_mask)

        im = masked_sm.plot(ax=ax, transform=ccrs.PlateCarree(), cmap="YlGnBu",
                            add_colorbar=False)

        ax.coastlines()
        ax.add_feature(cfeature.BORDERS, linestyle=':')
        ax.add_feature(cfeature.STATES, linewidth=0.5)

        if row_idx == 0:
            ax.set_title(f"{int(gap * 100)}% Gaps", fontsize=11)
        else:
            ax.set_title(f"", fontsize=11)
        if col_idx == 0:
            ax.text(-0.12, 0.5, title, va='center', ha='right',
                    fontsize=12, rotation=90, transform=ax.transAxes)

# Shared colorbar
cax = fig.add_subplot(gs[:, -1])
pos = cax.get_position()
cax.set_position([pos.x0, pos.y0 + pos.height * 0.15, pos.width, pos.height * 0.7])
cbar = fig.colorbar(im, cax=cax, orientation='vertical')
cbar.set_label("Soil Moisture")

# Save and show
plt.savefig("CYGNSS_gap_comparison_drought_flood.png", dpi=300, bbox_inches='tight')
plt.show()
```




```
masked_sm.shape
```

```
(1, 31, 32)
```

```
import xarray as xr
import numpy as np

# Get the original values
original = masked_sm

# Create a new lat coordinate by appending a dummy lat value
new_lat = np.append(original.lat.values, original.lat.values[-1] + (original.lat.values[-1] - original.lat.values[-2]))

# Create an empty (NaN-filled) array with new shape (1, 32, 32)
expanded_data = np.full((1, 32, 32), np.nan, dtype=np.float32)

# Copy existing data into the top 31 rows
expanded_data[:, :31, :] = original.values

# Create a new expanded DataArray
masked_sm_expanded = xr.DataArray(
    expanded_data,
    dims=["time", "lat", "lon"],
    coords={
        "time": original.time,
        "lat": new_lat,
        "lon": original.lon
    }
)
masked_sm_expanded.shape
```

```
(1, 32, 32)
```

```
img = masked_sm_expanded.values[0]
masks = np.array(create_mask(img))[np.newaxis]
incomplete_data = np.array([impute_data(img)])
print(incomplete_data.shape)
# # Combine the imputed data and the mask into a two-channel input
combined_data = np.concatenate((incomplete_data, masks), axis=0)[np.newaxis]
print(combined_data.shape)
```

```
(1, 32, 32)
(1, 2, 32, 32)
```

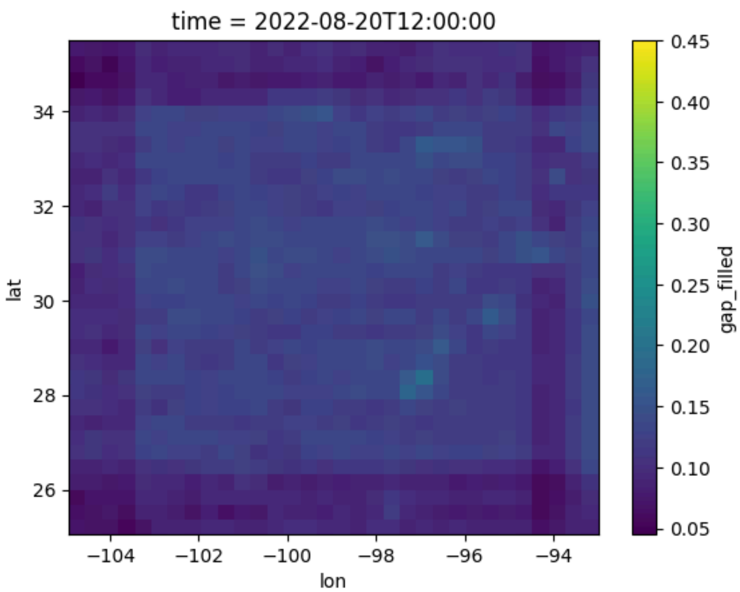
```
model(torch.from_numpy(combined_data).cuda()).cpu().detach().numpy()
```

```
array([[[[0.06985443, 0.06900196, 0.06646743, ..., 0.07475529,
          0.08914833, 0.08713016],
         [0.06458997, 0.06651377, 0.07236394, ..., 0.06712553,
          0.08601389, 0.09326524],
         [0.0569919 , 0.07010572, 0.07185911, ..., 0.06589624,
          0.08138721, 0.09637274],
         ...,
         [0.04539715, 0.06105171, 0.0588043 , ..., 0.06477017,
          0.08108994, 0.10902807],
         [0.05989096, 0.06617787, 0.05120999, ..., 0.07789353,
          0.08511012, 0.10678676],
         [0.07788891, 0.07498159, 0.07177931, ..., 0.07719763,
          0.08300406, 0.08783691]]]], dtype=float32)
```

```
masked_sm_expanded['gap_filled'] = (('time', 'lat', 'lon'), model(torch.from_numpy(combined_data).cuda()).cpu().detach().num
```

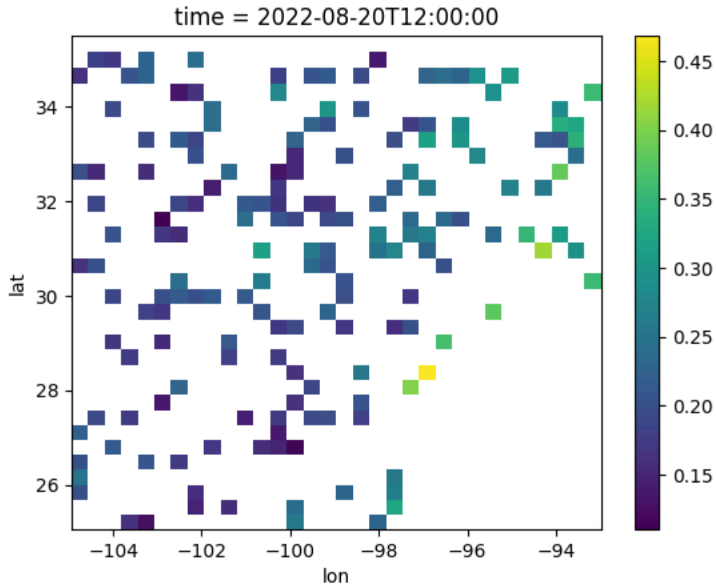
```
masked_sm_expanded['gap_filled'].plot(vmax=0.45)
```

```
<matplotlib.collections.QuadMesh at 0x78d95ab08790>
```




```
masked_sm_expanded.plot()
```

<matplotlib.collections.QuadMesh at 0x78d95a9e4390>

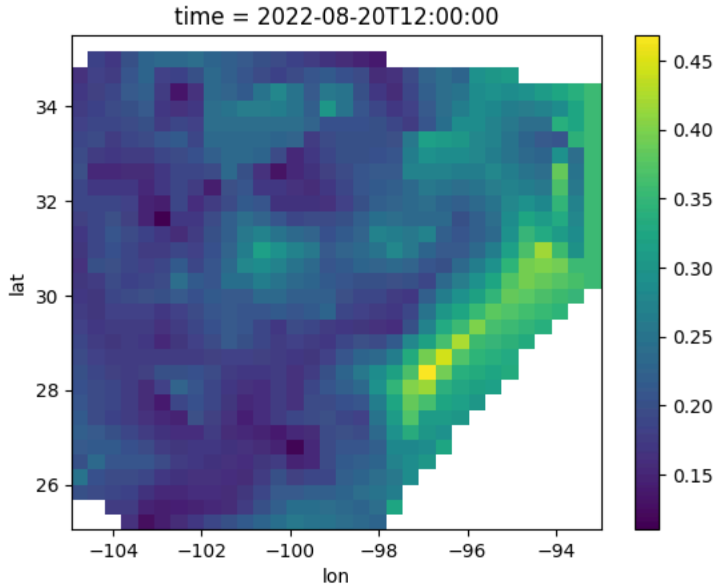



```
import numpy as np
import xarray as xr
from scipy.interpolate import griddata

def gap_fill_bilinear(data):
    # Flatten coordinates
    lon, lat = np.meshgrid(data.lon.values, data.lat.values)

    # Get valid points (non-NaN)
    valid_mask = ~np.isnan(data.values)
    points = np.column_stack((lon[valid_mask], lat[valid_mask]))
    values = data.values[valid_mask]

    # Create grid of all points
    grid_points = np.column_stack((lon.ravel(), lat.ravel()))

    # Bilinear interpolation (method='linear')
    filled_values = griddata(points, values, grid_points, method='linear')

    # Reshape to original grid
    filled_array = filled_values.reshape(data.shape)

    # Replace only NaNs in original with interpolated values
    filled_combined = np.where(np.isnan(data), filled_array, data)

    # Return new DataArray
    return xr.DataArray(
        filled_combined,
        coords=data.coords,
        dims=data.dims,
        name=f"{data.name}_filled" if data.name else None
    )
```

```
filled_sm = gap_fill_bilinear(masked_sm_expanded.isel(time=0))
```

```
filled_sm.plot()
```

```
<matplotlib.collections.QuadMesh at 0x78d97a5adcd0>
```



```
# import numpy as np
# import xarray as xr
# import matplotlib.pyplot as plt
# import cartopy.crs as ccrs
# import cartopy.feature as cfeature
# from matplotlib import gridspec
# import torch

# # Dummy CYGNSS-like data
# lat = np.linspace(25, 35, 32)
# lon = np.linspace(-105, -93, 32)
# time = np.array(["2022-07-15", "2022-08-20"], dtype="datetime64")

# # Create random data with NaNs
# np.random.seed(0)
# data = np.random.rand(2, 32, 32).astype(np.float32)
# mask_array = np.random.rand(32, 32) > 0.5
# data[:, ~mask_array] = np.nan

# # Construct dataset and mask
# ds_cygnss_gap_fill_final = xr.Dataset({
#     "SM_daily_gap_filled": (["time", "lat", "lon"], data)
# }, coords={"time": time, "lat": lat, "lon": lon})

# mask = xr.DataArray(mask_array.astype(int), dims=["lat", "lon"], coords={"lat": lat, "lon": lon})



# # Event and gap settings
# event_dict = {
#     "2022-07-15": "Drought (July 15)",
#     "2022-08-20": "Flood (Aug 20)"
# }
# gap_levels = [0.2, 0.4, 0.6, 0.8]

# # Create figure
# fig = plt.figure(figsize=(5 * len(gap_levels) + 1.5, 10))
# gs = gridspec.GridSpec(len(event_dict), len(gap_levels) + 1,
#                        width_ratios=[1]*len(gap_levels) + [0.05],
#                        hspace=0.15, wspace=0.05)

# for row_idx, (date_str, title) in enumerate(event_dict.items()):
#     for col_idx, gap in enumerate(gap_levels):
#         ax = fig.add_subplot(gs[row_idx, col_idx], projection=ccrs.PlateCarree())

#         sm_data = ds_cygnss_gap_fill_final.SM_daily_gap_filled.sel(time=date_str)
#         masked_sm = sm_data.where(mask == 1)

#         rng = np.random.default_rng(seed=100 * row_idx + col_idx)
#         gap_mask = rng.uniform(size=masked_sm.shape) > gap
#         masked_sm = masked_sm.where(gap_mask)

#         # Prepare input for model
#         img = masked_sm.values
#         masks = np.array(create_mask(img))[np.newaxis]
#         incomplete_data = np.array([impute_data(img)])
#         combined_data = np.concatenate((incomplete_data, masks), axis=0)[np.newaxis]

#         device = next(model.parameters()).device  # Detect if model is on GPU or CPU
#         input_tensor = torch.from_numpy(combined_data).float().to(device)
#         output = model(input_tensor).cpu().detach().numpy()[0, 0]

#         # # Model prediction
#         # output = model(torch.from_numpy(combined_data).float()).cpu().detach().numpy()[0, 0]

#         # Plot
#         lon_grid, lat_grid = np.meshgrid(masked_sm.lon.values, masked_sm.lat.values)
#         im = ax.pcolormesh(lon_grid, lat_grid, output, cmap="YlGnBu", shading="auto", transform=ccrs.PlateCarree())

#         ax.coastlines()
#         ax.add_feature(cfeature.BORDERS, linestyle=':')
#         ax.add_feature(cfeature.STATES, linewidth=0.5)

#         if row_idx == 0:
#             ax.set_title(f"{int(gap * 100)}% Gaps", fontsize=11)
#         if col_idx == 0:
#             ax.text(-0.12, 0.5, title, va='center', ha='right',
```

```
#                       fontsize=12, rotation=90, transform=ax.transAxes)

# # Colorbar
# cax = fig.add_subplot(gs[:, -1])
# pos = cax.get_position()
# cax.set_position([pos.x0, pos.y0 + pos.height * 0.15, pos.width, pos.height * 0.7])
# cbar = fig.colorbar(im, cax=cax, orientation='vertical')
# cbar.set_label("Soil Moisture")

# plt.savefig("CYGNSS_DL_gapfilled_comparison.png", dpi=300, bbox_inches='tight')
# plt.show()
```

```
import numpy as np
import xarray as xr
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from matplotlib import gridspec
import torch


# Event definitions
event_dict = {
    "2022-07-15": "Drought (July 15)",
    "2022-08-20": "Flood (Aug 20)"
}
gap_levels = [0.2, 0.4, 0.6, 0.8]

# Set up figure
fig = plt.figure(figsize=(5 * len(gap_levels) + 1.5, 10))
gs = gridspec.GridSpec(len(event_dict), len(gap_levels) + 1,
                       width_ratios=[1]*len(gap_levels) + [0.05],
                       hspace=0.15, wspace=0.05)

# Land overlay feature to mask oceans
ocean = cfeature.NaturalEarthFeature('physical', 'ocean', '10m',
                                    edgecolor='none', facecolor='white', zorder=10)

# Loop through events and gaps
for row_idx, (date_str, title) in enumerate(event_dict.items()):
    for col_idx, gap in enumerate(gap_levels):
        ax = fig.add_subplot(gs[row_idx, col_idx], projection=ccrs.PlateCarree())

        # Select and mask CYGNSS data
        sm_data = ds_cygnss_gap_fill_final.SM_daily_gap_filled.sel(
            lat=slice(obs.lat.values[0], obs.lat.values[-1]),
            lon=slice(obs.lon.values[0], obs.lon.values[-1]),
            time=date_str
        )
        masked_sm = sm_data.where(mask == 1)

        # Artificial gaps
        rng = np.random.default_rng(seed=100 * row_idx + col_idx)
        gap_mask = rng.uniform(size=masked_sm.shape) > gap
        masked_sm = masked_sm.where(gap_mask)

        # # Prepare model input
        # img = masked_sm.values
        # masks = np.array(create_mask(img))[np.newaxis]
        # incomplete_data = np.array([impute_data(img)])
        # combined_data = np.concatenate((incomplete_data, masks), axis=0)[np.newaxis]

        # Expand to 32×32 if needed
        img = masked_sm.values
        # Ensure shape is (1, 32, 32)
        if img.shape[1] != 32:
            expanded_img = np.full((1, 32, 32), np.nan, dtype=np.float32)
            expanded_img[:, :img.shape[1], :] = img

            # Also expand lat coordinate accordingly
            new_lat = np.append(masked_sm.lat.values,
                                masked_sm.lat.values[-1] + (masked_sm.lat.values[-1] - masked_sm.lat.values[-2]))

            masked_sm = xr.DataArray(
                expanded_img,
                dims=["time", "lat", "lon"],
                coords={
                    "time": masked_sm.time,
                    "lat": new_lat,
                    "lon": masked_sm.lon
                }
            )
            img = masked_sm.values

        # Prepare model input
        masks = np.array(create_mask(img[0]))[np.newaxis]
        incomplete_data = np.array([impute_data(img[0])])
        combined_data = np.concatenate((incomplete_data, masks), axis=0)[np.newaxis]

```

```
        # Run model
        device = next(model.parameters()).device
        input_tensor = torch.from_numpy(combined_data).float().to(device)
        output = model(input_tensor).cpu().detach().numpy()[0, 0]

        # Plot prediction
        lon_grid, lat_grid = np.meshgrid(masked_sm.lon.values, masked_sm.lat.values)
        im = ax.pcolormesh(lon_grid, lat_grid, output, cmap="YlGnBu", shading="auto", transform=ccrs.PlateCarree())

        # Map features
        ax.coastlines()
        ax.add_feature(cfeature.BORDERS, linestyle=':')
        ax.add_feature(cfeature.STATES, linewidth=0.5)
        ax.add_feature(ocean)  # visually mask ocean

        if row_idx == 0:
            ax.set_title(f"{int(gap * 100)}% Gaps", fontsize=11)
        if col_idx == 0:
            ax.text(-0.12, 0.5, title, va='center', ha='right',
                    fontsize=12, rotation=90, transform=ax.transAxes)

# Colorbar
cax = fig.add_subplot(gs[:, -1])
pos = cax.get_position()
cax.set_position([pos.x0, pos.y0 + pos.height * 0.15, pos.width, pos.height * 0.7])
cbar = fig.colorbar(im, cax=cax, orientation='vertical')
cbar.set_label("Soil Moisture")

plt.savefig("CYGNSS_DL_gapfilled_comparison.png", dpi=300, bbox_inches='tight')
plt.show()
```