

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

Mounted at /content/drive

```
1 !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) (0.12)
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) (4.22.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy) (1.3.1)
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) (3.1.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy) (2.9.0)
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>=3.6->cartopy) (1.16.0)
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
```

```
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
```

```
1 import xarray as xr
2 !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.
```

```
1 import xarray as xr
2
3 vars = ['SM_daily', 'latitude', 'longitude']
4
5 ds_cygnss = xr.open_mfdataset('CYGNSS/data2022/ucar_cu_cygnss_sm_v1_2022_???.dap.nc')[vars].compute()
6 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_???'

```

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)

```

1 # Adding latitude and longitude as 1D coordinates
2 lat_data = ds_cygnss['latitude'].isel(time=0, lon=0).values
3 lon_data = ds_cygnss['longitude'].isel(time=0, lat=0).values
4
5 ds_cygnss_ = ds_cygnss.assign_coords(lat=lat_data, lon=lon_data)
6 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

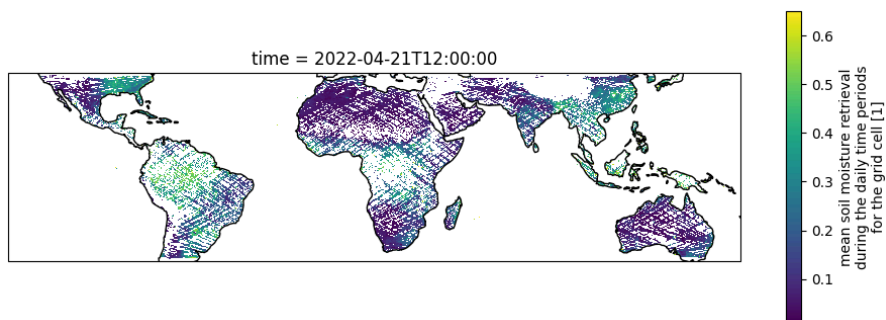
► Indexes: (3)

► Attributes: (45)

```

1 import matplotlib.pyplot as plt
2 import cartopy.crs as ccrs
3 import cartopy.feature as cfeature
4
5 fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(11.69, 4), subplot_kw={'projection': ccrs.PlateCarree()})
6 ds_cygnss_.SM_daily.isel(time=110).plot(ax=ax, cmap='viridis')
7 ax.coastlines()
8 plt.savefig('fig1_gaps.png', dpi=500)

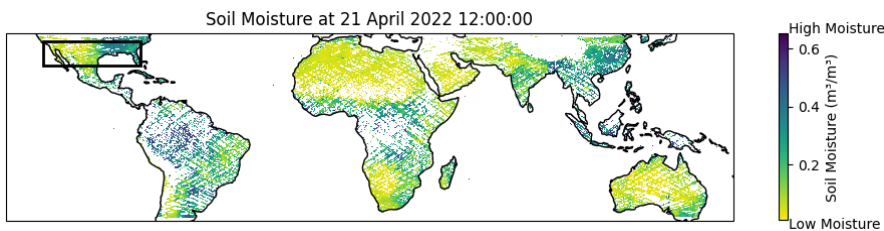
```



```

1 import matplotlib.pyplot as plt
2 import cartopy.crs as ccrs
3 import cartopy.feature as cfeature
4 from matplotlib.patches import Rectangle
5
6 # Create the plot with Cartopy projection
7 fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(11.69, 4), subplot_kw={'projection': ccrs.PlateCarree()})
8 sm_plot = ds_cygnss_.SM_daily.isel(time=110).plot(ax=ax, cmap='viridis_r', cbar_kwargs={'shrink': 0.6, 'label': 'Soil Moistu
9
10 # Add coastlines
11 ax.coastlines()
12
13 # Add title
14 ax.set_title('Soil Moisture at 21 April 2022 12:00:00')
15
16 # Add black box (lat = slice(25, 35), lon = slice(-120, -90))
17 lon_min, lon_max = -120, -80
18 lat_min, lat_max = 25, 35
19
20 # Create a Rectangle for the box (black border, no fill)
21 box = Rectangle((lon_min, lat_min), lon_max - lon_min, lat_max - lat_min,
22                 linewidth=2, edgecolor='black', facecolor='none', transform=ccrs.PlateCarree())
23
24 # Add the rectangle to the plot
25 ax.add_patch(box)
26
27 # Modify colorbar and add text
28 cbar = sm_plot.colorbar
29 cbar.set_label('Soil Moisture (m³/m³)', fontsize=10)
30 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')
31
32 # Save the figure
33 plt.savefig('fig1_gaps_1.png', dpi=500)

```



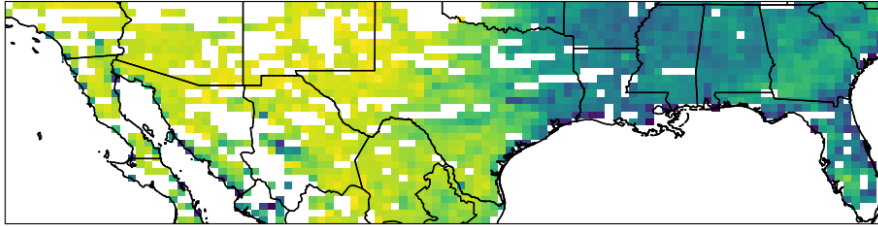
```

1 import matplotlib.pyplot as plt
2 import cartopy.crs as ccrs
3 import cartopy.feature as cfeature
4
5 fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(11.69, 8), subplot_kw={'projection': ccrs.PlateCarree()})
6 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
7
8 ax.coastlines()
9 ax.add_feature(cfeature.STATES.with_scale('10m')) # Add state borders
10 ax.add_feature(cfeature.BORDERS, linestyle=':')
11 # ax.set_title('Soil Moisture at 21 April 2022 12:00:00')
12
13 # # Adding text on the colorbar (near its label)
14 # cbar = sm_plot.colorbar
15 # cbar.set_label('Soil Moisture (m³/m³)', fontsize=10)
16 # 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')
17
18 plt.savefig('fig1_gaps_2.png', dpi=500)

```



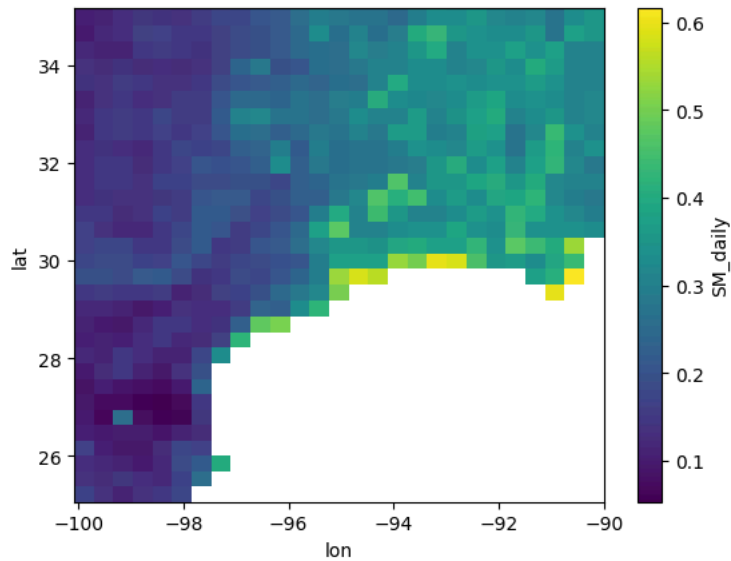
time = 2022-04-21T12:00:00



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



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




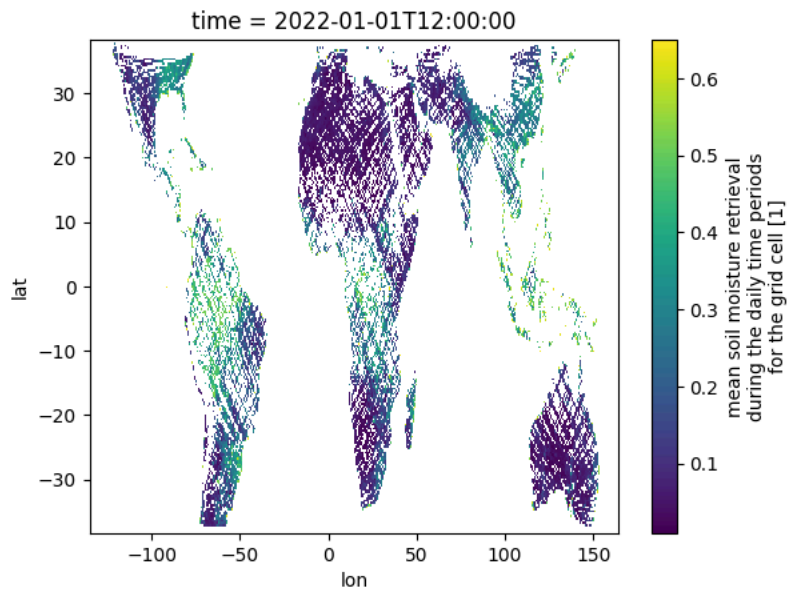
```
1 ds_cygnss_tx = ds_cygnss_.sel(lat=slice(25,35)).sel(lon=slice(-100,-90)).SM_daily
2 ds_cygnss_tx = ds_cygnss_.SM_daily
3 ds_cygnss_tx.shape
```



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

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

 <matplotlib.collections.QuadMesh at 0x78d95afaca50>



✓ Make sub images

```

1 import numpy as np
2 import xarray as xr
3 import matplotlib.pyplot as plt
4 from skimage.util import view_as_blocks
5
6 def slice_data_for_blocks(data, block_size=(64, 64)):
7     """
8     Slice the data to ensure dimensions are divisible by the block size.
9
10    Parameters:
11    data (xarray.DataArray): The original data array.
12    block_size (tuple): The size of the blocks (sub-images).
13
14    Returns:
15    xarray.DataArray: The sliced data array.
16    tuple: The shape of the cropped data array.
17    """
18    lat_size, lon_size = data.shape[-2], data.shape[-1]
19    lat_new = (lat_size // block_size[0]) * block_size[0]
20    lon_new = (lon_size // block_size[1]) * block_size[1]
21    data_sliced = data.isel(lat=slice(0, lat_new), lon=slice(0, lon_new))
22    return data_sliced, (lat_new, lon_new)
23
24 def generate_sub_images(data, block_size=(64, 64)):
25     """
26     Generate sub-images of the specified block size from the original image.
27
28    Parameters:
29    data (numpy array): The original image array of shape (H, W).
30    block_size (tuple): The size of the blocks (sub-images) to generate.
31
32    Returns:
33    numpy array: An array of sub-images.
34    """
35    H, W = data.shape
36    blocks = view_as_blocks(data, block_shape=block_size)
37    sub_images = blocks.reshape(-1, *block_size)
38    return sub_images
39
40 def recreate_image_from_sub_images(sub_images, original_shape, block_size=(64, 64)):
41     """
42     Recreate the original image from sub-images.
43
44    Parameters:
45    sub_images (numpy array): An array of sub-images.
46    original_shape (tuple): The shape of the cropped image (H, W).
47    block_size (tuple): The size of the blocks (sub-images) used.
48
49    Returns:
50    numpy array: The recreated original image.
51    """
52    H, W = original_shape
53    h_blocks = H // block_size[0]
54    w_blocks = W // block_size[1]
55    sub_images = sub_images.reshape(h_blocks, w_blocks, *block_size)
56    recreated_image = np.block([[sub_images[i, j] for j in range(w_blocks)] for i in range(h_blocks)])
57    return recreated_image
58
59 # Ensure data dimensions are divisible by block size
60 data_sliced, cropped_shape = slice_data_for_blocks(ds_cygnss_tx, block_size=(8, 8))
61
62 # Generate sub-images
63 sub_images = generate_sub_images(data_sliced[0,:,:].values, block_size=(8, 8))
64
65
66
67 # Recreate the original image
68 recreated_image = recreate_image_from_sub_images(sub_images, cropped_shape, block_size=(8, 8))
69
70 print(f"Original slice shape: {ds_cygnss_tx.shape}")
71 print(f"Sliced data shape: {data_sliced.shape}")
72 print(f"Sub-images shape: {sub_images.shape}")
73 print(f"Recreated image shape: {recreated_image.shape}")
74
75 # Visualization
76 fig, ax = plt.subplots(1, 2, figsize=(12, 6))
77

```



```

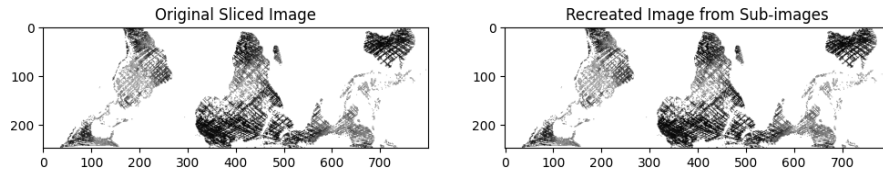
78 ax[0].imshow(data_sliced[0,:,:], cmap='gray')
79 ax[0].set_title('Original Sliced Image')
80
81 ax[1].imshow(recreated_image, cmap='gray')
82 ax[1].set_title('Recreated Image from Sub-images')
83
84 plt.show()

```

```

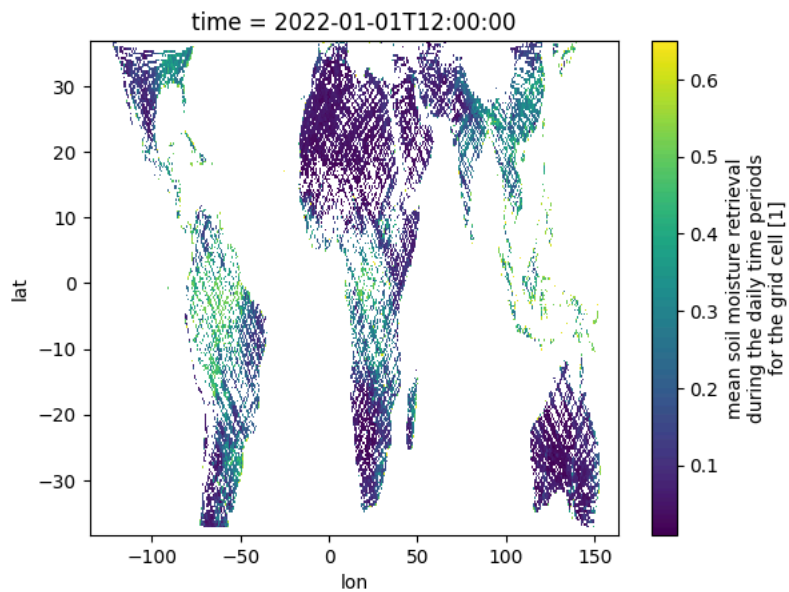
Original slice shape: (360, 252, 802)
Sliced data shape: (360, 248, 800)
Sub-images shape: (3100, 8, 8)
Recreated image shape: (248, 800)

```



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

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



```

1 import numpy as np
2 import xarray as xr
3 import matplotlib.pyplot as plt
4 from skimage.util import view_as_blocks
5
6 def slice_data_for_blocks(data, block_size=(64, 64)):
7     """
8     Slice the data to ensure dimensions are divisible by the block size.
9
10    Parameters:
11    data (xarray.DataArray): The original data array.
12    block_size (tuple): The size of the blocks (sub-images).
13
14    Returns:
15    xarray.DataArray: The sliced data array.
16    tuple: The shape of the cropped data array.
17    """
18    lat_size, lon_size = data.shape[-2], data.shape[-1]
19    lat_new = (lat_size // block_size[0]) * block_size[0]
20    lon_new = (lon_size // block_size[1]) * block_size[1]
21    data_sliced = data.isel(lat=slice(0, lat_new), lon=slice(0, lon_new))
22    return data_sliced, (lat_new, lon_new)
23
24 def generate_sub_images(data, block_size=(64, 64)):
25     """
26     Generate sub-images of the specified block size from the original image.
27
28    Parameters:
29    data (numpy array): The original image array of shape (H, W).
30    block_size (tuple): The size of the blocks (sub-images) to generate.
31
32    Returns:
33    numpy array: An array of sub-images.
34    """
35    H, W = data.shape
36    blocks = view_as_blocks(data, block_shape=block_size)
37    sub_images = blocks.reshape(-1, *block_size)
38    return sub_images
39
40 def recreate_image_from_sub_images(sub_images, original_shape, block_size=(64, 64)):
41     """
42     Recreate the original image from sub-images.
43
44    Parameters:
45    sub_images (numpy array): An array of sub-images.
46    original_shape (tuple): The shape of the cropped image (H, W).
47    block_size (tuple): The size of the blocks (sub-images) used.
48
49    Returns:
50    numpy array: The recreated original image.
51    """
52    H, W = original_shape
53    h_blocks = H // block_size[0]
54    w_blocks = W // block_size[1]
55    sub_images = sub_images.reshape(h_blocks, w_blocks, *block_size)
56    recreated_image = np.block([[sub_images[i, j] for j in range(w_blocks)] for i in range(h_blocks)])
57    return recreated_image
58
59 def visualize_sub_images(sub_images, block_size=(64, 64), ncols=8):
60     """
61     Visualize the sub-images.
62
63    Parameters:
64    sub_images (numpy array): An array of sub-images.
65    block_size (tuple): The size of the blocks (sub-images).
66    ncols (int): The number of columns for visualization.
67    """
68    nrows = int(np.ceil(len(sub_images) / ncols))
69    fig, axes = plt.subplots(nrows, ncols, figsize=(11, 8))
70    for ax, sub_image in zip(axes.flat, sub_images):
71        ax.imshow(sub_image, cmap='gray')
72        ax.axis('off')
73    for ax in axes.flat[len(sub_images):]:
74        ax.axis('off')
75    plt.savefig('sub_images_before_training.png', dpi=500)
76
77 data_array = ds_cygnss_tx #ds_combined.isel(time=times_to_isel).Band1

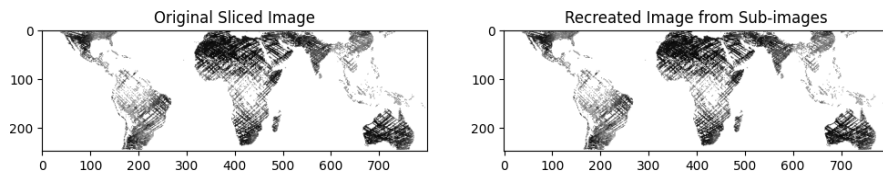
```

```

78 # Select a specific time slice
79 data_slice = ds_cygnss_tx #data_array.sel(time=data_array.time[0])
80
81 # Ensure data dimensions are divisible by block size
82 data_sliced, cropped_shape = slice_data_for_blocks(data_slice, block_size=(8, 8))
83
84 # Convert to numpy and flip along vertical axis to correct orientation
85 data_sliced_np = np.flipud(data_sliced.values)
86
87 # Generate sub-images
88 sub_images = generate_sub_images(data_sliced_np[0,:,:], block_size=(8, 8))
89
90 # Recreate the original image
91 recreated_image = recreate_image_from_sub_images(sub_images, cropped_shape, block_size=(8, 8))
92
93 # Flip the recreated image back to the original orientation
94 recreated_image = np.flipud(recreated_image)
95
96 print(f"Original slice shape: {data_slice.shape}")
97 print(f"Sliced data shape: {data_sliced.shape}")
98 print(f"Sub-images shape: {sub_images.shape}")
99 print(f"Recreated image shape: {recreated_image.shape}")
100
101 # Visualization
102 fig, ax = plt.subplots(1, 2, figsize=(12, 6))
103
104 ax[0].imshow(np.flipud(data_sliced_np[0,:,:]), cmap='gray')
105 ax[0].set_title('Original Sliced Image')
106
107 ax[1].imshow(recreated_image, cmap='gray')
108 ax[1].set_title('Recreated Image from Sub-images')
109
110 plt.show()
111
112 # # Visualize sub-images
113 # visualize_sub_images(sub_images, block_size=(64, 64), ncols=8)

```

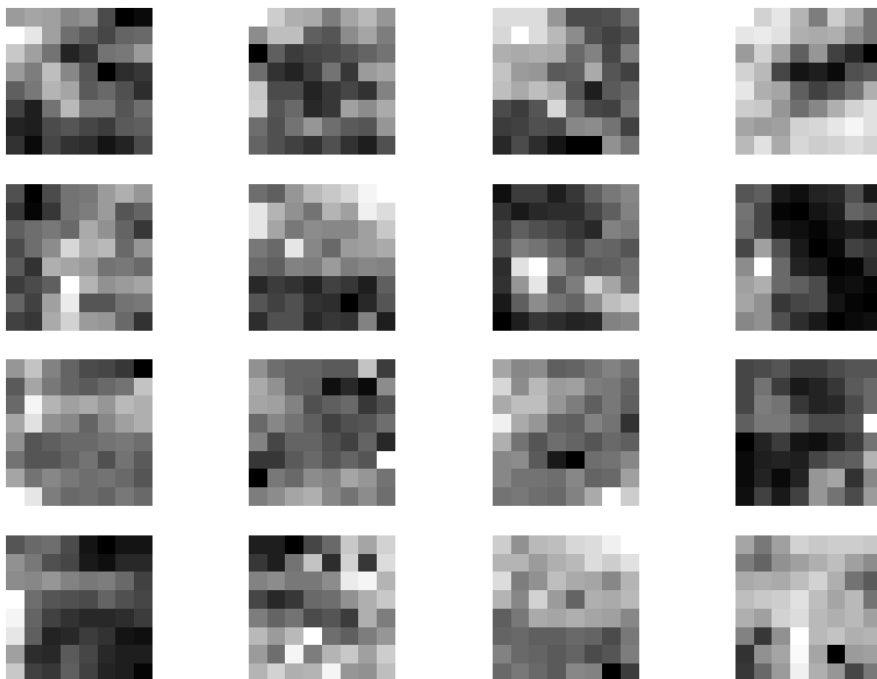
Original slice shape: (360, 252, 802)
 Sliced data shape: (360, 248, 800)
 Sub-images shape: (3100, 8, 8)
 Recreated image shape: (248, 800)



```

1 def remove_nan_sub_images(sub_images):
2     """
3     Remove sub-images that contain NaN values.
4
5     Parameters:
6     sub_images (numpy array): An array of sub-images.
7
8     Returns:
9     numpy array: Filtered array of sub-images without NaN values.
10    """
11    mask = ~np.isnan(sub_images).any(axis=(1, 2))
12    return sub_images[mask]
13
14 # Assuming ds_combined is already loaded and times_to_isel is defined
15 data_array = ds_cygnss_tx # ds_combined.isel(time=times_to_isel).Band1
16
17 # Process each time slice
18 all_sub_images = []
19 for time in data_array.time:
20     data_slice = data_array.sel(time=time)
21
22     # Ensure data dimensions are divisible by block size
23     data_sliced, cropped_shape = slice_data_for_blocks(data_slice, block_size=(8, 8))
24
25     # Convert to numpy and flip along vertical axis to correct orientation
26     data_sliced_np = np.flipud(data_sliced.values)
27
28     # Generate sub-images
29     sub_images = generate_sub_images(data_sliced_np, block_size=(8, 8))
30
31     # Remove sub-images with NaN values
32     sub_images = remove_nan_sub_images(sub_images)
33
34     all_sub_images.append(sub_images)
35
36 # Combine all sub-images into a single array for visualization if needed
37 all_sub_images = np.concatenate(all_sub_images, axis=0)
38
39
40 print(all_sub_images.shape, np.min(all_sub_images), np.max(all_sub_images))
41
42 (1178, 8, 8) 0.010150299 0.60524344
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```



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

✓ Gap Filling Starts

```
1 import torch
2 import torch.nn as nn
3 import torch.optim as optim
4 from torch.utils.data import Dataset, DataLoader
5 from torchvision import transforms
6 import numpy as np
7 import matplotlib.pyplot as plt
```

```

1 # Define the U-Net architecture
2 class UNet(nn.Module):
3     def __init__(self, in_channels, out_channels):
4         super(UNet, self).__init__()
5
6         def conv_block(in_channels, out_channels):
7             block = nn.Sequential(
8                 nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
9                 nn.ReLU(inplace=True),
10                 nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1),
11                 nn.ReLU(inplace=True)
12             )
13             return block
14
15         self.encoder1 = conv_block(in_channels, 64)
16         self.encoder2 = conv_block(64, 128)
17         self.encoder3 = conv_block(128, 256)
18         self.encoder4 = conv_block(256, 512)
19
20         self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
21
22         self.middle = conv_block(512, 1024)
23
24         self.upconv4 = nn.ConvTranspose2d(1024, 512, kernel_size=2, stride=2)
25         self.decoder4 = conv_block(1024, 512)
26         self.upconv3 = nn.ConvTranspose2d(512, 256, kernel_size=2, stride=2)
27         self.decoder3 = conv_block(512, 256)
28         self.upconv2 = nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2)
29         self.decoder2 = conv_block(256, 128)
30         self.upconv1 = nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2)
31         self.decoder1 = conv_block(128, 64)
32
33         self.final_conv = nn.Conv2d(64, out_channels, kernel_size=1)
34
35     def forward(self, x):
36         enc1 = self.encoder1(x)
37         enc2 = self.encoder2(self.pool(enc1))
38         enc3 = self.encoder3(self.pool(enc2))
39         enc4 = self.encoder4(self.pool(enc3))
40
41         middle = self.middle(self.pool(enc4))
42
43         dec4 = self.upconv4(middle)
44         dec4 = torch.cat((dec4, enc4), dim=1)
45         dec4 = self.decoder4(dec4)
46
47         dec3 = self.upconv3(dec4)
48         dec3 = torch.cat((dec3, enc3), dim=1)
49         dec3 = self.decoder3(dec3)
50
51         dec2 = self.upconv2(dec3)
52         dec2 = torch.cat((dec2, enc2), dim=1)
53         dec2 = self.decoder2(dec2)
54
55         dec1 = self.upconv1(dec2)
56         dec1 = torch.cat((dec1, enc1), dim=1)
57         dec1 = self.decoder1(dec1)
58
59         out = self.final_conv(dec1)
60         return out
61
62 class FSRCNN(nn.Module):
63     def __init__(self, d=56, s=12, m=4, upscale_factor=1):
64         super(FSRCNN, self).__init__()
65         self.first_part = nn.Sequential(
66             nn.Conv2d(2, d, kernel_size=5, padding=5//2),
67             nn.PReLU(d)
68         )
69
70         self.mid_parts = [nn.Sequential(
71             nn.Conv2d(d, s, kernel_size=1),
72             nn.PReLU(s)
73         )]
74
75         for _ in range(m - 1):
76             self.mid_parts.append(nn.Sequential(
77                 nn.Conv2d(s, s, kernel_size=3, padding=3//2),

```

```

78         nn.PReLU(s)
79     ))
80
81     self.mid_parts = nn.Sequential(*self.mid_parts)
82
83     self.last_part = nn.Sequential(
84         nn.Conv2d(s, d, kernel_size=1),
85         nn.PReLU(d),
86         nn.ConvTranspose2d(d, 1, kernel_size=9, stride=upscale_factor, padding=9//2, output_padding=upscale_factor-1)
87     )
88
89     def forward(self, x):
90         x = self.first_part(x)
91         x = self.mid_parts(x)
92         x = self.last_part(x)
93         return x

1 # Define the custom dataset
2 class CYGNSSDataset(Dataset):
3     def __init__(self, incomplete_data, complete_data, transform=None):
4         self.incomplete_data = incomplete_data
5         self.complete_data = complete_data
6         self.transform = transform
7
8     def __len__(self):
9         return len(self.incomplete_data)
10
11     def __getitem__(self, idx):
12         incomplete = self.incomplete_data[idx]
13         complete = self.complete_data[idx]
14
15         if self.transform:
16             incomplete = self.transform(incomplete)
17             complete = self.transform(complete)
18
19         return incomplete, complete
20
21 # Masking function to simulate missing values
22 def mask_data(data, missing_percentage):
23     masked_data = data.copy()
24     mask = np.random.rand(*data.shape) < missing_percentage
25     masked_data[mask] = np.nan # Use NaN for missing values
26     return masked_data
27
28 # Create a mask channel where 1 indicates valid data and 0 indicates missing data
29 def create_mask(data):
30     mask = ~np.isnan(data)
31     return mask.astype(np.float32)
32
33 # Impute NaN values with the mean of the non-NaN values
34 def impute_data(data):
35     nan_mask = np.isnan(data)
36     mean_value = np.nanmean(data)
37     data[nan_mask] = mean_value
38     return data

1 all_sub_images[:, np.newaxis, :, :].shape
↵ (1178, 1, 8, 8)

1 print(all_sub_images.shape, np.min(all_sub_images), np.max(all_sub_images))
↵ (1178, 8, 8) 0.010150299 0.60524344

1 import numpy as np
2 from sklearn.model_selection import train_test_split
3 from sklearn.preprocessing import MinMaxScaler
4 from torch.utils.data import DataLoader, Dataset, Subset
5 import joblib
6
7 complete_data = all_sub_images[:, np.newaxis, :, :]
8
9 # Save the scaler parameters to disk
10 scaler_params = {'min': np.min(all_sub_images), 'max': np.max(all_sub_images)}
11 # scaler_filename = "/content/drive/MyDrive/EarthEngineExperiments/1/scaler_params.joblib"

```

```

11 # scaler_filename = /content/drive/mydrive/EdiEngineExports/ + scaler_params.joblib
12 # joblib.dump(scaler_params, scaler_filename)
13
14 # # Load the scaler parameters from disk (for demonstration purposes)
15 # loaded_params = joblib.load(scaler_filename)
16 # loaded_min = loaded_params['min']
17 # loaded_max = loaded_params['max']
18
19 # Manually apply min-max scaling using the loaded parameters
20 scaled_complete_data = 2 * (complete_data - scaler_params['min']) / (scaler_params['max'] - scaler_params['min'])
21 scaled_complete_data = scaled_complete_data

```

```
1 scaled_complete_data.shape
```

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

```

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

```

```

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

```

```
1 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 ]
```

```
1 # Create the dataset
2 dataset = CYGNSSDataset(combined_data, complete_data)
3
4 # Define the split ratios
5 train_ratio = 0.7
6 val_ratio = 0.15
7 test_ratio = 0.15
8
9 # Split the dataset indices
10 train_size = int(train_ratio * len(dataset))
11 val_size = int(val_ratio * len(dataset))
12 test_size = len(dataset) - train_size - val_size
13
14 train_indices, temp_indices = train_test_split(range(len(dataset)), train_size=train_size, shuffle=True)
15 val_indices, test_indices = train_test_split(temp_indices, test_size=test_size, shuffle=True)
16
17 # Create subset datasets
18 train_dataset = Subset(dataset, train_indices)
19 val_dataset = Subset(dataset, val_indices)
20 test_dataset = Subset(dataset, test_indices)
21
22 # Create DataLoaders
23 batch_size = 4
24 train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
25 val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
26 test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
27
28 print(f"Train dataset size: {len(train_dataset)}")
29 print(f"Validation dataset size: {len(val_dataset)}")
30 print(f"Test dataset size: {len(test_dataset)}")
```

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

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

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

```
↗ (2, 8, 8) (1178, 2, 8, 8)
```

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

```

1 import torch
2 import torch.nn as nn
3 import torch.optim as optim
4
5 # Define DataLoader for validation data
6 val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
7
8 # Training the model
9 num_epochs = 1000
10 best_val_loss = float('inf')
11 best_model_path = "/content/drive/MyDrive/EarthEngineExports/" + "best_model_cygnss_gap_fill.pth"
12 train = False
13 if train:
14     for epoch in range(num_epochs):
15         model.train()
16         for i, (incomplete, complete) in enumerate(train_loader):
17             incomplete = incomplete.float().cuda()
18             complete = complete.float().cuda()
19
20             # Forward pass
21             outputs = model(incomplete)
22             loss = criterion(outputs, complete)
23
24             # Backward pass and optimization
25             optimizer.zero_grad()
26             loss.backward()
27             optimizer.step()
28
29             if (i + 1) % 10 == 0:
30                 print(f'Epoch [{epoch+1}/{num_epochs}], Step [{i+1}/{len(train_loader)}], Loss: {loss.item():.4f}')
31
32         # Evaluate on validation data
33         model.eval()
34         val_loss = 0.0
35         with torch.no_grad():
36             for incomplete, complete in val_loader:
37                 incomplete = incomplete.float().cuda()
38                 complete = complete.float().cuda()
39
40                 outputs = model(incomplete)
41                 loss = criterion(outputs, complete)
42                 val_loss += loss.item()
43
44         val_loss /= len(val_loader)
45         print(f'Epoch [{epoch+1}/{num_epochs}], Validation Loss: {val_loss:.4f}')
46
47         # Save the model if validation loss has decreased
48         if val_loss < best_val_loss:
49             best_val_loss = val_loss
50             torch.save(model.state_dict(), best_model_path)
51             print(f'Saved best model with validation loss: {best_val_loss:.4f}')
52
53 # Load the best model for further use or evaluation
54 model.load_state_dict(torch.load(best_model_path))
55 print('Loaded best model for further use or evaluation')

```

 Loaded best model for further use or evaluation

```

1 import numpy as np
2 from sklearn.metrics import r2_score
3 import torch
4
5 # Evaluation loop with R-squared calculation
6 def evaluate_model_with_r2(model, dataloader, criterion):
7     model.eval()
8     eval_loss = 0.0
9     all_targets = []
10    all_outputs = []
11
12    with torch.no_grad():
13        for inputs, targets in dataloader:
14            # Convert inputs and targets to tensors and move to GPU
15            inputs = torch.tensor(inputs).float().cuda()
16            targets = torch.tensor(targets).float().cuda()
17
18            # Ensure inputs have the correct number of channels
19            if inputs.shape[1] == 1: # If input has 1 channel, expand it to 2 channels
20                inputs = inputs.repeat(1, 2, 1, 1)
21
22            outputs = model(inputs)
23            loss = criterion(outputs, targets)
24
25            eval_loss += loss.item() * inputs.size(0)
26
27            all_targets.append(targets.cpu().numpy())
28            all_outputs.append(outputs.cpu().numpy())
29
30    eval_loss /= len(dataloader.dataset)
31    print(f'Evaluation Loss: {eval_loss:.4f}')
32
33    # Concatenate all batches
34    all_targets = np.concatenate(all_targets, axis=0).reshape(-1)
35    all_outputs = np.concatenate(all_outputs, axis=0).reshape(-1)
36
37    r2 = r2_score(all_targets, all_outputs)
38    print(f'R-squared: {r2:.4f}')
39    return eval_loss, r2
40
41 # Assuming test_loader is already defined
42 test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
43
44 # Instantiate and run the evaluation
45 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)

```

```

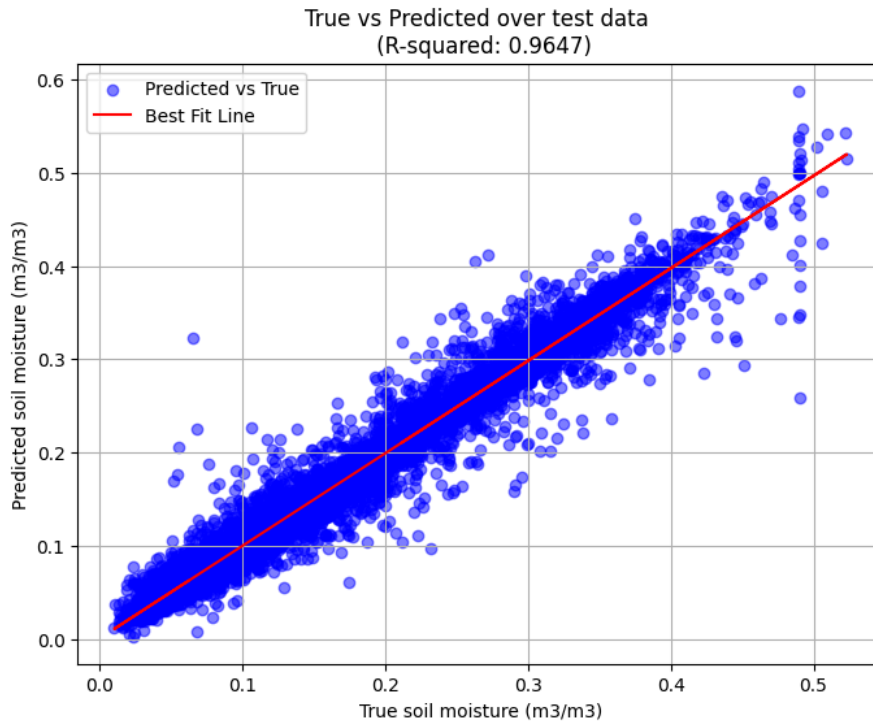
1 import matplotlib.pyplot as plt
2 import numpy as np
3 from sklearn.metrics import r2_score
4 import torch
5
6 # Evaluation loop with R-squared calculation and plotting
7 def evaluate_model_with_r2_and_plots(model, dataloader, criterion):
8     model.eval()
9     eval_loss = 0.0
10    all_targets = []
11    all_outputs = []
12
13    with torch.no_grad():
14        for inputs, targets in dataloader:
15            # Convert inputs and targets to tensors and move to GPU
16            inputs = torch.tensor(inputs).float().cuda()
17            targets = torch.tensor(targets).float().cuda()
18
19            # Ensure inputs have the correct number of channels
20            if inputs.shape[1] == 1: # If input has 1 channel, expand it to 2 channels
21                inputs = inputs.repeat(1, 2, 1, 1)
22
23            outputs = model(inputs)
24            loss = criterion(outputs, targets)
25
26            eval_loss += loss.item() * inputs.size(0)
27
28            all_targets.append(targets.cpu().numpy())
29            all_outputs.append(outputs.cpu().numpy())
30
31    eval_loss /= len(dataloader.dataset)
32    print(f'Evaluation Loss: {eval_loss:.4f}')
33
34    # Concatenate all batches
35    all_targets = np.concatenate(all_targets, axis=0).reshape(-1)
36    all_outputs = np.concatenate(all_outputs, axis=0).reshape(-1)
37
38    # Calculate R-squared
39    r2 = r2_score(all_targets, all_outputs)
40    print(f'R-squared: {r2:.4f}')
41
42    # Plot scatter plot with best fit line
43    plot_results(all_targets, all_outputs)
44
45    return eval_loss, r2
46
47 # Function to create scatter plot with best fit line
48 def plot_results(true_values, predicted_values):
49     plt.figure(figsize=(8, 6))
50
51     # Scatter plot
52     plt.scatter(true_values, predicted_values, color='blue', label='Predicted vs True', alpha=0.5)
53
54     # Best fit line
55     best_fit_line = np.poly1d(np.polyfit(true_values, predicted_values, 1))
56     plt.plot(true_values, best_fit_line(true_values), color='red', label='Best Fit Line')
57
58     # Plot labels
59     plt.xlabel('True soil moisture (m3/m3)')
60     plt.ylabel('Predicted soil moisture (m3/m3)')
61     plt.title('True vs Predicted over test data \n (R-squared: 0.9647)')
62     plt.legend()
63
64     # Show plot
65     plt.grid(True)
66     plt.savefig('best_fit.png', dpi=500)
67
68 # Assuming test_loader is already defined
69 test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
70
71 # Instantiate and run the evaluation with plots
72 evaluate_model_with_r2_and_plots(model, test_loader, criterion)
73

```

```

/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)

```



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

Generate Predictions

```

1 incomplete_data = ds_cygnss_tx.isel(time=110).data
2 incomplete_data.shape

```

```
(252, 802)
```

```

1 incomplete_data = ds_cygnss_tx.isel(time=110).data[:248, :800]
2 print(incomplete_data[np.newaxis].shape, np.sum(np.isnan(incomplete_data[np.newaxis])))
3
4 scaled_incomplete_data = 2 * (incomplete_data - scaler_params['min']) / (scaler_params['max'] - scaler_params['min'])
5 scaled_incomplete_data = scaled_incomplete_data#.compute()
6
7 print(scaled_incomplete_data.shape)
8 # Create mask and impute data
9 masks = np.array([create_mask(img) for img in scaled_incomplete_data])
10 incomplete_data = np.array([impute_data(img) for img in scaled_incomplete_data])
11 print(masks.shape)
12 # Combine the imputed data and the mask into a two-channel input
13 combined_data = np.concatenate((incomplete_data[np.newaxis], masks[np.newaxis]), axis=0)
14 print(combined_data.shape)
15
16 gap_filled = model(torch.tensor(combined_data[np.newaxis]).float().cuda()).cpu().detach().numpy()[0,0]
17 gap_filled.shape

```

```

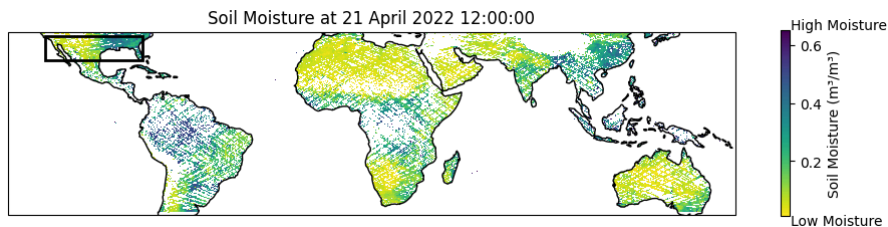
(1, 248, 800) 166519
(248, 800)
(248, 800)
(2, 248, 800)
/tmp/ipython-input-172-1112879747.py:36: RuntimeWarning: Mean of empty slice
mean_value = np.nanmean(data)
(248, 800)

```

```

1 import matplotlib.pyplot as plt
2 import cartopy.crs as ccrs
3 import cartopy.feature as cfeature
4 from matplotlib.patches import Rectangle
5
6 # Create the plot with Cartopy projection
7 fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(11.69, 4), subplot_kw={'projection': ccrs.PlateCarree()})
8 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
9
10 # Add coastlines
11 ax.coastlines()
12
13 # Add title
14 ax.set_title('Soil Moisture at 21 April 2022 12:00:00')
15
16 # Add black box (lat = slice(25, 35), lon = slice(-120, -90))
17 lon_min, lon_max = -120, -80
18 lat_min, lat_max = 25, 35
19
20 # Create a Rectangle for the box (black border, no fill)
21 box = Rectangle((lon_min, lat_min), lon_max - lon_min, lat_max - lat_min,
22                 linewidth=2, edgecolor='black', facecolor='none', transform=ccrs.PlateCarree())
23
24 # Add the rectangle to the plot
25 ax.add_patch(box)
26
27 # Modify colorbar and add text
28 cbar = sm_plot.colorbar
29 cbar.set_label('Soil Moisture (m³/m³)', fontsize=10)
30 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')
31
32 # Save the figure
33 plt.savefig('fig1_gaps_1.png', dpi=500)

```



```

1 ds_cygnss_gap_fill = ds_cygnss_tx.isel(time=110).isel(lat=slice(0,248)).isel(lon=slice(0,800))
2 ds_cygnss_gap_fill['SM_daily_gap_filled'] = (('lat', 'lon'), gap_filled)
3 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

```

1 mask = ds_cygnss_.SM_daily.mean(dim='time').isel(lat=slice(0,248)).isel(lon=slice(0,800)).values
2 mask = mask/mask
3 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)

```

```

1 ds_cygnss_gap_fill_mask = ds_cygnss_gap_fill.SM_daily_gap_filled*mask
2 ds_cygnss_gap_fill_mask = ds_cygnss_gap_fill_mask*(scaler_params['max'] - scaler_params['min'])/2 + scaler_params['min']
3 ds_cygnss_gap_fill_mask = ((ds_cygnss_gap_fill_mask - ds_cygnss_gap_fill_mask.mean())/ds_cygnss_gap_fill_mask.std()) *ds_cygnss_gap_fill_mask
4 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	

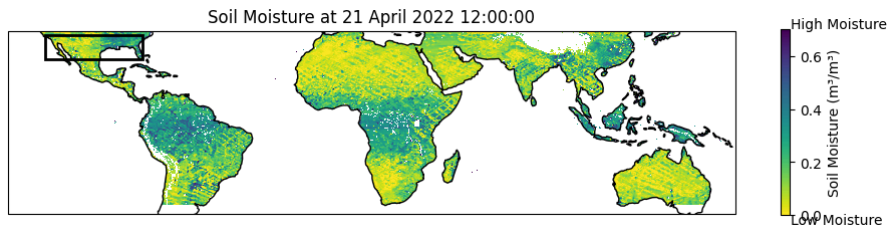
► Indexes: (2)

► Attributes: (0)

```

1 import matplotlib.pyplot as plt
2 import cartopy.crs as ccrs
3 import cartopy.feature as cfeature
4 from matplotlib.patches import Rectangle
5
6 # Create the plot with Cartopy projection
7 fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(11.69, 4), subplot_kw={'projection': ccrs.PlateCarree()})
8 sm_plot = ds_cygnss_gap_fill_mask.plot(ax=ax, cmap='viridis_r', cbar_kwargs={'shrink': 0.6, 'label': 'Soil Moisture (m³/m³)'})
9
10 # Add coastlines
11 ax.coastlines()
12
13 # Add title
14 ax.set_title('Soil Moisture at 21 April 2022 12:00:00')
15
16 # Add black box (lat = slice(25, 35), lon = slice(-120, -90))
17 lon_min, lon_max = -120, -80
18 lat_min, lat_max = 25, 35
19
20 # Create a Rectangle for the box (black border, no fill)
21 box = Rectangle((lon_min, lat_min), lon_max - lon_min, lat_max - lat_min,
22                 linewidth=2, edgecolor='black', facecolor='none', transform=ccrs.PlateCarree())
23
24 # Add the rectangle to the plot
25 ax.add_patch(box)
26
27 # Modify colorbar and add text
28 cbar = sm_plot.colorbar
29 cbar.set_label('Soil Moisture (m³/m³)', fontsize=10)
30 cbar.ax.text(1.05, 0.5, 'High Moisture\n\n\n\n\n\n\n\n\n\nLow Moisture', transform=cbar.ax.transAxes, va='center')
31
32 # Save the figure
33 plt.savefig('fig4_gap_filled_1.png', dpi=500)

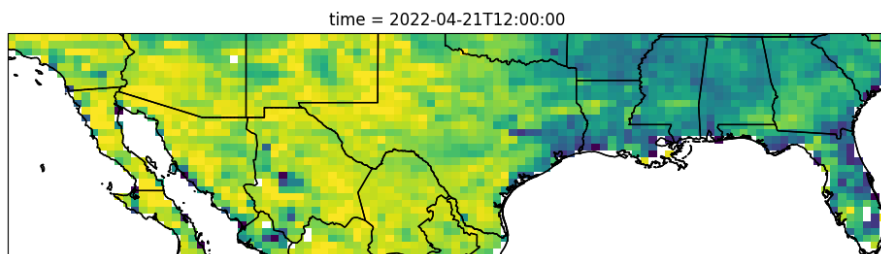
```



```

1 import matplotlib.pyplot as plt
2 import cartopy.crs as ccrs
3 import cartopy.feature as cfeature
4
5 fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(11.69, 8), subplot_kw={'projection': ccrs.PlateCarree()})
6 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
7
8 ax.coastlines()
9 ax.add_feature(cfeature.STATES.with_scale('10m')) # Add state borders
10 ax.add_feature(cfeature.BORDERS, linestyle=':')
11
12 plt.savefig('fig4_gap_filled_2.png', dpi=500)

```




```

1 import numpy as np
2 import xarray as xr
3 import torch
4 from tqdm import tqdm
5
6 # Helper functions (reuse from before)
7 def create_mask(data):
8     return (~np.isnan(data)).astype(np.float32)
9
10 def impute_data(data):
11     data = data.copy()
12     nan_mask = np.isnan(data)
13     data[nan_mask] = np.nanmean(data)
14     return data
15
16 # Subset CYGNSS data (adjust region if needed)
17 lat_slice = slice(0, 248)
18 lon_slice = slice(0, 800)
19 ds_crop = ds_cygnss_tx.isel(lat=lat_slice, lon=lon_slice)
20
21 # Initialize empty list for results
22 gap_filled_all = []
23
24 # Loop over time dimension
25 for t in tqdm(range(ds_crop.shape[0]), desc="Gap-filling over time"):
26     # Extract soil moisture at time t
27     incomplete_data = ds_crop.isel(time=t).values
28
29     # Skip if fully missing
30     if np.isnan(incomplete_data).all():
31         gap_filled_all.append(np.full_like(incomplete_data, np.nan))
32         continue
33
34     # Scale
35     scaled = 2 * (incomplete_data - scaler_params['min']) / (scaler_params['max'] - scaler_params['min'])
36
37     # Mask and impute
38     mask = create_mask(scaled)
39     scaled = impute_data(scaled)
40
41     # Stack as 2-channel input
42     combined = np.stack([scaled, mask], axis=0) # Shape: [2, H, W]
43
44     # Model prediction
45     with torch.no_grad():
46         input_tensor = torch.tensor(combined[np.newaxis]).float().cuda() # Shape: [1, 2, H, W]
47         output_tensor = model(input_tensor).cpu().numpy()[0, 0] # Shape: [H, W]
48
49     # Append output
50     gap_filled_all.append(output_tensor)
51
52 # Convert to numpy array
53 gap_filled_all = np.array(gap_filled_all) # Shape: [time, lat, lon]
54
55 # Create xarray DataArray
56 ds_gap_filled = xr.DataArray(
57     gap_filled_all,
58     dims=['time', 'lat', 'lon'],
59     coords={
60         'time': ds_crop.time.values,
61         'lat': ds_crop.lat.values,
62         'lon': ds_crop.lon.values
63     },
64     name="SM_daily_gap_filled"
65 )
66
67 # Optional: save to NetCDF
68 ds_gap_filled.to_netcdf("cygnss_gap_filled_all.nc")
69 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'

```

1 import xarray as xr
2 import numpy as np
3
4 # Subset and crop the data (same region as deep learning gap-fill)
5 lat_slice = slice(0, 248)
6 lon_slice = slice(0, 800)
7 ds_interp_input = ds_cygnss_tx.isel(lat=lat_slice, lon=lon_slice)
8
9 # Apply linear interpolation along the time dimension
10 ds_linear_interp = ds_interp_input.interpolate_na(
11     dim="time",
12     method="linear",
13     fill_value="extrapolate" # Optional: extrapolates at edges
14 )
15
16 # Rename for consistency
17 ds_linear_interp.name = "SM_daily_linear_interp"
18
19 # Save the result
20 ds_linear_interp.to_netcdf("cygnss_gap_filled_linear.nc")
21 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'

```
1 !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)

1 Start coding or generate with AI.

```

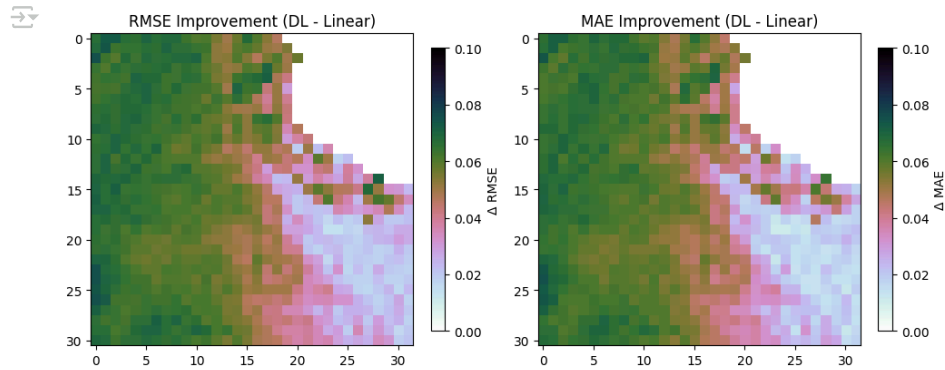
1 import numpy as np
2 import xarray as xr
3 from sklearn.metrics import mean_squared_error, mean_absolute_error
4 from skimage.metrics import structural_similarity as ssim
5 import matplotlib.pyplot as plt
6
7 # Define the region of interest (Texas in this case)
8 lat_slice = slice(25, 35)
9 lon_slice = slice(-105, -93)
10
11 # Extract observed (original), DL gap-filled, and Linear interpolated datasets
12 obs = ds_cygnss_tx.sel(lat=lat_slice, lon=lon_slice)
13 dl = ds_gap_filled.sel(lat=lat_slice, lon=lon_slice)
14 li = ds_linear_interp.sel(lat=lat_slice, lon=lon_slice)
15
16 # Align all three over time
17 obs, dl, li = xr.align(obs, dl, li)
18
19 # Initialize metric maps
20 lat_size = obs.sizes['lat']
21 lon_size = obs.sizes['lon']
22
23 rmse_dl_map = np.full((lat_size, lon_size), np.nan)
24 mae_dl_map = np.full((lat_size, lon_size), np.nan)
25 ssim_dl_map = np.full((lat_size, lon_size), np.nan)
26
27 rmse_li_map = np.full((lat_size, lon_size), np.nan)
28 mae_li_map = np.full((lat_size, lon_size), np.nan)
29 ssim_li_map = np.full((lat_size, lon_size), np.nan)
30
31 # Loop over each grid point
32 for i in range(lat_size):
33     for j in range(lon_size):
34         obs_series = obs[:, i, j].values
35         dl_series = dl[:, i, j].values
36         li_series = li[:, i, j].values
37
38         # Only evaluate at times where observations exist
39         mask = ~np.isnan(obs_series)
40         if np.sum(mask) < 10:
41             continue
42
43         obs_valid = obs_series[mask]
44         dl_valid = dl_series[mask]
45         li_valid = li_series[mask]
46
47         # Skip if DL or LI has NaNs (could happen at edge or masked regions)
48         if np.isnan(dl_valid).any() or np.isnan(li_valid).any():
49             continue
50
51         # DL metrics
52         rmse_dl_map[i, j] = np.sqrt(mean_squared_error(obs_valid, dl_valid))
53         mae_dl_map[i, j] = mean_absolute_error(obs_valid, dl_valid)
54         try:
55             ssim_dl_map[i, j] = ssim(obs_valid.reshape(-1, 1), dl_valid.reshape(-1, 1), data_range=obs_valid.max() - obs_val
56         except:
57             pass
58
59         # LI metrics
60         rmse_li_map[i, j] = np.sqrt(mean_squared_error(obs_valid, li_valid))
61         mae_li_map[i, j] = mean_absolute_error(obs_valid, li_valid)
62         try:
63             ssim_li_map[i, j] = ssim(obs_valid.reshape(-1, 1), li_valid.reshape(-1, 1), data_range=obs_valid.max() - obs_val
64         except:
65             pass

```

```

1 import matplotlib.pyplot as plt
2
3 fig, axs = plt.subplots(1, 2, figsize=(10, 5))
4
5 # RMSE improvement (DL - Linear)
6 diff_rmse = (rmse_li_map - rmse_dl_map) * -1 # Positive = DL is better
7 im1 = axs[0].imshow(diff_rmse, cmap='cubehelix_r', vmin=0.0, vmax=0.1)
8 axs[0].set_title('RMSE Improvement (DL - Linear)')
9 plt.colorbar(im1, ax=axs[0], fraction=0.04, label='Δ RMSE')
10
11 # MAE improvement (DL - Linear)
12 diff_mae = (mae_li_map - mae_dl_map) * -1
13 im2 = axs[1].imshow(diff_mae, cmap='cubehelix_r', vmin=0.0, vmax=0.1)
14 axs[1].set_title('MAE Improvement (DL - Linear)')
15 plt.colorbar(im2, ax=axs[1], fraction=0.04, label='Δ MAE')
16
17 plt.tight_layout()
18 plt.savefig("fig7_dl_vs_linear_vs_observed_metrics.png", dpi=500)

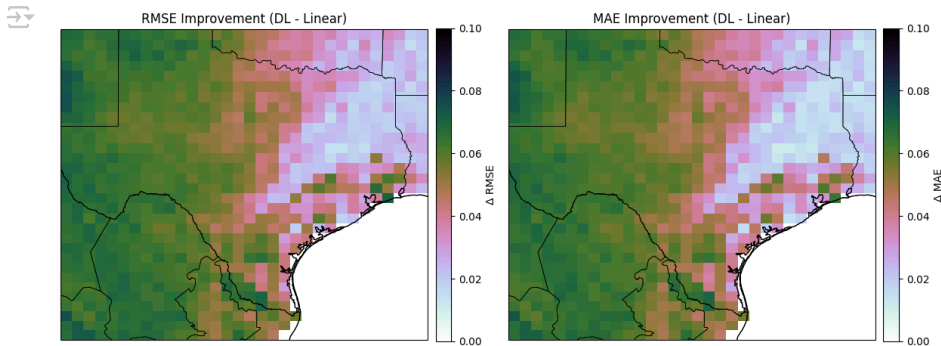
```



```

1 lat = obs.lat.values
2 lon = obs.lon.values
3
4 # Now construct the xarray DataArrays using lat/lon
5 da_rmse = xr.DataArray(diff_rmse, coords=[('lat', lat), ('lon', lon)])
6 da_mae = xr.DataArray(diff_mae, coords=[('lat', lat), ('lon', lon)])
7
8 import xarray as xr
9 import matplotlib.pyplot as plt
10 import cartopy.crs as ccrs
11 import cartopy.feature as cfeature
12
13 # Get lat/lon from obs grid
14 lat = obs.lat.values
15 lon = obs.lon.values
16
17 # Compute improvement
18 diff_rmse = (rmse_li_map - rmse_dl_map) * -1
19 diff_mae = (mae_li_map - mae_dl_map) * -1
20
21 # Wrap into xarray
22 da_rmse = xr.DataArray(diff_rmse, coords=[('lat', lat), ('lon', lon)])
23 da_mae = xr.DataArray(diff_mae, coords=[('lat', lat), ('lon', lon)])
24
25 # Plot
26 fig, axs = plt.subplots(1, 2, figsize=(12, 6), subplot_kw={'projection': ccrs.PlateCarree()})
27
28 im1 = da_rmse.plot(ax=axs[0], cmap='cubehelix_r', vmin=0.0, vmax=0.1, add_colorbar=False)
29 axs[0].set_title('RMSE Improvement (DL - Linear)')
30 axs[0].coastlines()
31 axs[0].add_feature(cfeature.STATES, linewidth=0.5)
32 axs[0].add_feature(cfeature.BORDERS, linestyle=':')
33
34 im2 = da_mae.plot(ax=axs[1], cmap='cubehelix_r', vmin=0.0, vmax=0.1, add_colorbar=False)
35 axs[1].set_title('MAE Improvement (DL - Linear)')
36 axs[1].coastlines()
37 axs[1].add_feature(cfeature.STATES, linewidth=0.5)
38 axs[1].add_feature(cfeature.BORDERS, linestyle=':')
39
40 fig.colorbar(im1, ax=axs[0], orientation='vertical', fraction=0.04, pad=0.02, label='Δ RMSE')
41 fig.colorbar(im2, ax=axs[1], orientation='vertical', fraction=0.04, pad=0.02, label='Δ MAE')
42
43 plt.tight_layout()
44 plt.savefig("fig7_dl_vs_linear_vs_observed_metrics.png", dpi=500, bbox_inches='tight')
45 plt.show()

```



```

1 # Central Texas drought grid (near Travis County)
2 lat_drought, lon_drought = 30.3, -97.8
3
4 # DFW flood grid (near Dallas)
5 lat_flood, lon_flood = 32.8, -96.8

```

```

1 # Dates to visualize
2 event_dates = {
3     "Drought_July15": "2022-07-15",
4     "Flood_Aug22": "2022-08-22"
5 }

```

```
1 !pip install cartopy
```

```

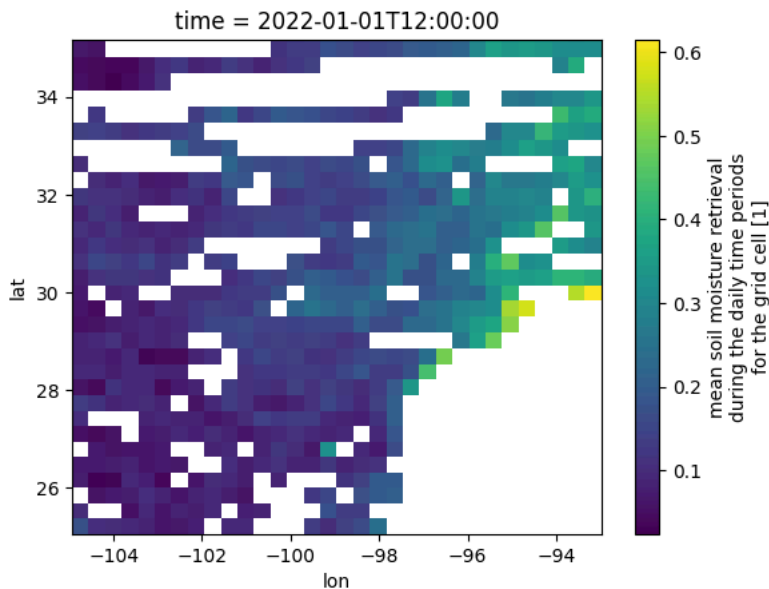
Requirement already satisfied: cartopy in /usr/local/lib/python3.11/dist-packages (0.24.1)
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/python3.11/dist-packages (from cartopy) (2.0.2)
Requirement already satisfied: matplotlib>=3.6 in /usr/local/lib/python3.11/dist-packages (from cartopy) (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) (1.2.1)
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) (4.22.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy) (1.4.5)
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) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.6->cartopy) (2.8.2)
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>=3.6) (1.16.0)

```

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

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

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



```

1 mask = obs.mean(dim='time').values
2 mask = mask/mask
3 mask

```



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, ..., nan,
           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)
```

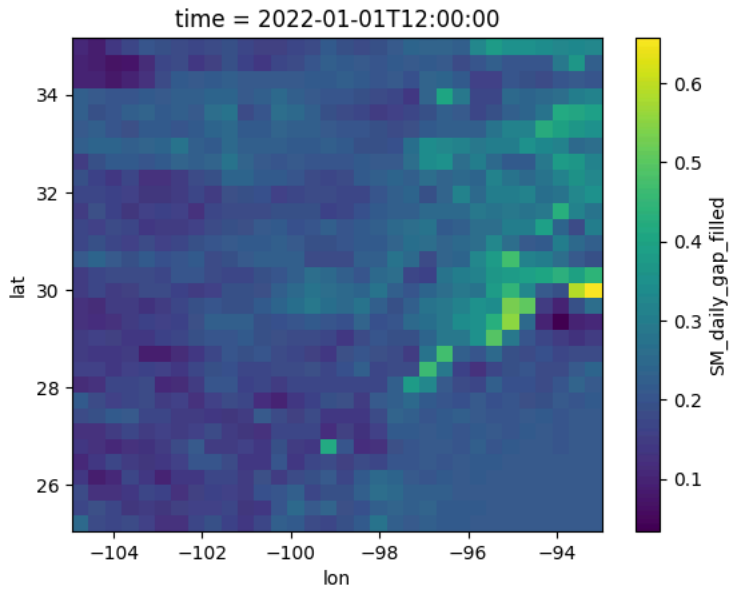
time	(time)	datetime64[ns]	2022-01-01T12:00:00 ... 2022-12-...
lat	(lat)	float32	25.22 25.53 25.84 ... 34.65 34.99
lon	(lon)	float32	-104.8 -104.4 ... -93.55 -93.17

▼ Attributes:

comment :	units represent soil moisture content as a fractional volume (cm ³ 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.v
```

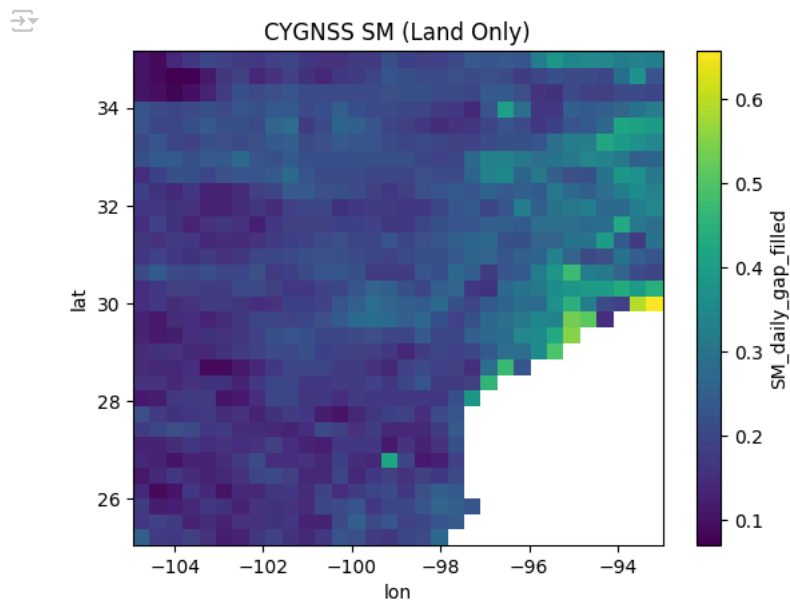

 <matplotlib.collections.QuadMesh at 0x78d95af713d0>



```

1 import xarray as xr
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # Extract the first time step
6 sm_data = ds_cygnss_gap_fill_final.SM_daily_gap_filled.sel(
7     lat=slice(obs.lat.values[0], obs.lat.values[-1]),
8     lon=slice(obs.lon.values[0], obs.lon.values[-1])
9 ).isel(time=0)
10
11 # Assume you have land_mask as 2D (lat, lon), same shape as sm_data
12 # Apply the mask: set ocean (mask==0) to NaN
13 masked_sm = sm_data.where(mask == 1)
14
15 # Plot
16 masked_sm.plot()
17 plt.title("CYGNSS SM (Land Only)")
18 plt.show()


```



```

1 import cartopy.crs as ccrs
2 import cartopy.feature as cfeature
3 import matplotlib.pyplot as plt
4
5 # Define event dates (can be extended)
6 event_dict = {
7     "Drought_July15": "2022-07-15",
8     "Flood_Aug20": "2022-08-20",
9     "Drought_Sept01": "2022-09-01"
10 }
11
12 # Directory to save figures
13 output_dir = "./cygnss_event_maps/"
14 import os
15 os.makedirs(output_dir, exist_ok=True)
16
17 for event_name, date_str in event_dict.items():
18     # Get data
19     sm_data = ds_cygnss_gap_fill_final.SM_daily_gap_filled.sel(
20         lat=slice(obs.lat.values[0], obs.lat.values[-1]),
21         lon=slice(obs.lon.values[0], obs.lon.values[-1]),
22         time=date_str
23     )
24
25     # Apply land mask
26     masked_sm = sm_data.where(mask == 1)
27
28     # Plot with Cartopy
29     fig = plt.figure(figsize=(10, 6))
30     ax = plt.axes(projection=ccrs.PlateCarree())
31     im = masked_sm.plot(ax=ax, transform=ccrs.PlateCarree(), cmap="YlGnBu", cbar_kwargs={"label": "Soil Moisture"})
32
33     # Add features
34     ax.coastlines()
35     ax.add_feature(cfeature.BORDERS, linestyle=':')
36     ax.add_feature(cfeature.STATES, linewidth=0.5)
37     ax.set_title(f"CYGNSS SM during {event_name.replace('_', ' ')}", fontsize=14)
38
39     # Save
40     filepath = os.path.join(output_dir, f"{event_name}.png")
41     plt.savefig(filepath, dpi=300, bbox_inches='tight')
42     plt.close()
43
44     print(f"Saved: {filepath}")

```

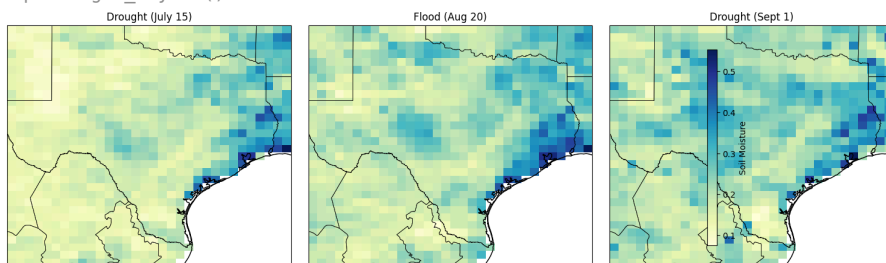
 Saved: ./cygnss_event_maps/Drought_July15.png
 Saved: ./cygnss_event_maps/Flood_Aug20.png
 Saved: ./cygnss_event_maps/Drought_Sept01.png

```

1 import cartopy.crs as ccrs
2 import cartopy.feature as cfeature
3 import matplotlib.pyplot as plt
4
5 # Define event dates and titles
6 event_dict = {
7     "2022-07-15": "Drought (July 15)",
8     "2022-08-20": "Flood (Aug 20)",
9     "2022-09-01": "Drought (Sept 1)"
10 }
11
12 # Create figure
13 n_events = len(event_dict)
14 fig, axs = plt.subplots(1, n_events, figsize=(5 * n_events, 6),
15                         subplot_kw={'projection': ccrs.PlateCarree()})
16
17 if n_events == 1:
18     axs = [axs] # ensure it's iterable if only one subplot
19
20 # Loop over events
21 for ax, (date_str, title) in zip(axs, event_dict.items()):
22     sm_data = ds_cygnss_gap_fill_final.SM_daily_gap_filled.sel(
23         lat=slice(obs.lat.values[0], obs.lat.values[-1]),
24         lon=slice(obs.lon.values[0], obs.lon.values[-1]),
25         time=date_str
26     )
27     masked_sm = sm_data.where(mask == 1)
28
29
30     im = masked_sm.plot(ax=ax, transform=ccrs.PlateCarree(), cmap="YlGnBu",
31                        add_colorbar=False) # suppress colorbar per plot
32     ax.coastlines()
33     ax.add_feature(cfeature.BORDERS, linestyle=':')
34     ax.add_feature(cfeature.STATES, linewidth=0.5)
35     ax.set_title(f"{title}", fontsize=12)
36
37 # Add a shared colorbar
38 cbar = fig.colorbar(im, ax=axs, orientation='vertical', shrink=0.7, label="Soil Moisture")
39
40 # Save and show
41 plt.tight_layout()
42 plt.savefig("CYGNSS_SM_Events_2022.png", dpi=300)
43 plt.show()

```

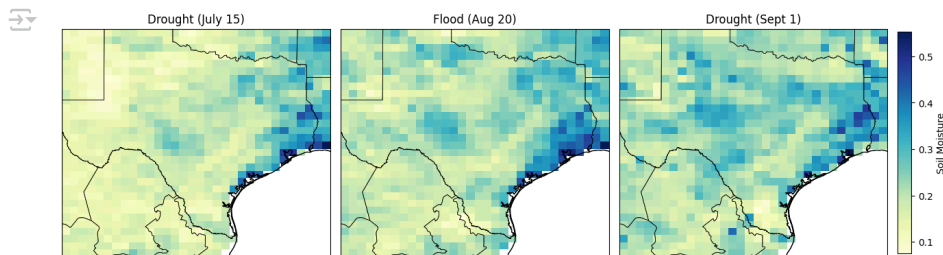
 /tmp/ipython-input-212-476118328.py:41: UserWarning: This figure includes Axes t
plt.tight_layout()



```

1 import cartopy.crs as ccrs
2 import cartopy.feature as cfeature
3 import matplotlib.pyplot as plt
4 from matplotlib import gridspec
5
6 # Define event dates and titles
7 event_dict = {
8     "2022-07-15": "Drought (July 15)",
9     "2022-08-20": "Flood (Aug 20)",
10    "2022-09-01": "Drought (Sept 1)"
11 }
12
13 # Set up figure with space on right for colorbar
14 n_events = len(event_dict)
15 fig = plt.figure(figsize=(5 * n_events + 1, 6))
16 gs = gridspec.GridSpec(1, n_events + 1, width_ratios=[1]*n_events + [0.05], wspace=0.05)
17
18 axs = []
19 for i in range(n_events):
20     ax = fig.add_subplot(gs[0, i], projection=ccrs.PlateCarree())
21     axs.append(ax)
22
23 # Plot each event
24 for ax, (date_str, title) in zip(axs, event_dict.items()):
25     sm_data = ds_cygnss_gap_fill_final.SM_daily_gap_filled.sel(
26         lat=slice(obs.lat.values[0], obs.lat.values[-1]),
27         lon=slice(obs.lon.values[0], obs.lon.values[-1]),
28         time=date_str
29     )
30     masked_sm = sm_data.where(mask == 1)
31
32     im = masked_sm.plot(ax=ax, transform=ccrs.PlateCarree(), cmap="YlGnBu",
33                        add_colorbar=False)
34     ax.coastlines()
35     ax.add_feature(cfeature.BORDERS, linestyle=':')
36     ax.add_feature(cfeature.STATES, linewidth=0.5)
37     ax.set_title(title, fontsize=12)
38
39 # Add vertical colorbar with shorter height
40 cax = fig.add_subplot(gs[0, -1])
41 pos = cax.get_position()
42 # Shrink to 70% of original height and center
43 cax.set_position([pos.x0, pos.y0 + pos.height * 0.15, pos.width, pos.height * 0.7])
44
45 cbar = fig.colorbar(im, cax=cax, orientation='vertical')
46 cbar.set_label("Soil Moisture")
47
48 # Save and show
49 plt.savefig("CYGNSS_SM_Events_2022.png", dpi=300, bbox_inches='tight')
50 plt.show()

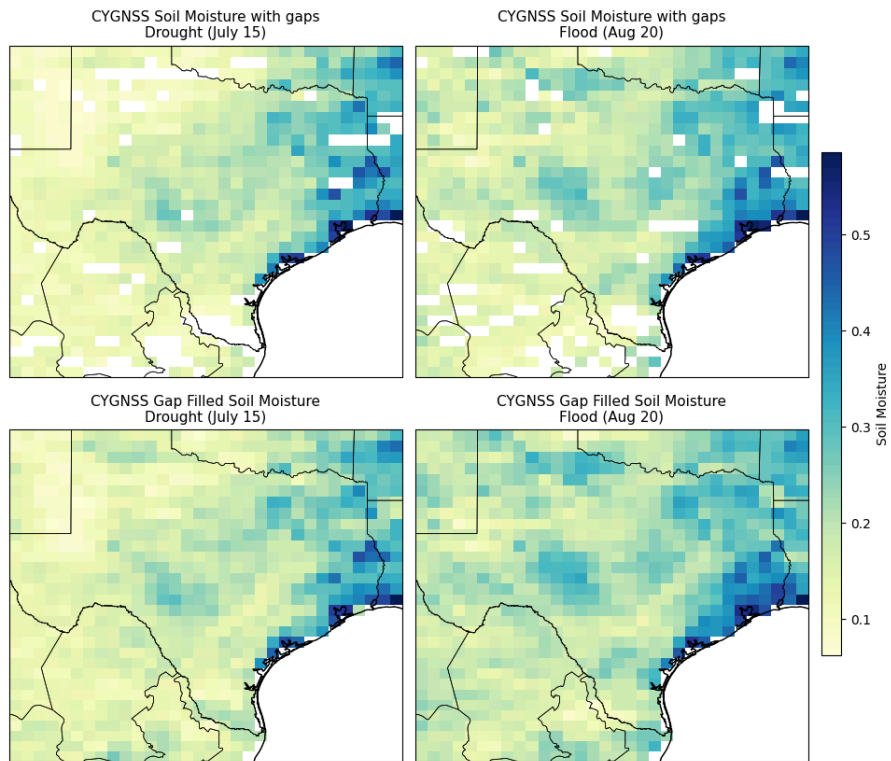
```



```

1 import cartopy.crs as ccrs
2 import cartopy.feature as cfeature
3 import matplotlib.pyplot as plt
4 from matplotlib import gridspec
5
6 # Define event dates and titles
7 event_dict = {
8     "2022-07-15": "Drought (July 15)",
9     "2022-08-20": "Flood (Aug 20)"
10 }
11
12 n_events = len(event_dict)
13
14 # Create 2-row grid: Row 1 = OBS, Row 2 = CYGNSS
15 fig = plt.figure(figsize=(5 * n_events + 1.5, 10))
16 gs = gridspec.GridSpec(2, n_events + 1, width_ratios=[1]*n_events + [0.05], hspace=0.15, wspace=0.05)
17
18 axs_obs = []
19 axs_dl = []
20
21 # Plot observations (top row)
22 for i, (date_str, title) in enumerate(event_dict.items()):
23     ax = fig.add_subplot(gs[0, i], projection=ccrs.PlateCarree())
24     axs_obs.append(ax)
25
26     obs_data = obs.sel(time=date_str)
27     im_obs = obs_data.plot(ax=ax, transform=ccrs.PlateCarree(), cmap="YlGnBu",
28                             add_colorbar=False)
29
30     ax.coastlines()
31     ax.add_feature(cfeature.BORDERS, linestyle=':')
32     ax.add_feature(cfeature.STATES, linewidth=0.5)
33     ax.set_title(f"CYGNSS Soil Moisture with gaps\n{title}", fontsize=11)
34
35 # Plot CYGNSS (bottom row)
36 for i, (date_str, title) in enumerate(event_dict.items()):
37     ax = fig.add_subplot(gs[1, i], projection=ccrs.PlateCarree())
38     axs_dl.append(ax)
39
40     sm_data = ds_cygnss_gap_fill_final.SM_daily_gap_filled.sel(
41         lat=slice(obs.lat.values[0], obs.lat.values[-1]),
42         lon=slice(obs.lon.values[0], obs.lon.values[-1]),
43         time=date_str
44     )
45     masked_sm = sm_data.where(mask == 1)
46
47     im_dl = masked_sm.plot(ax=ax, transform=ccrs.PlateCarree(), cmap="YlGnBu",
48                             add_colorbar=False)
49
50     ax.coastlines()
51     ax.add_feature(cfeature.BORDERS, linestyle=':')
52     ax.add_feature(cfeature.STATES, linewidth=0.5)
53     ax.set_title(f"CYGNSS Gap Filled Soil Moisture\n{title}", fontsize=11)
54
55 # Colorbar outside
56 cax = fig.add_subplot(gs[:, -1])
57 pos = cax.get_position()
58 cax.set_position([pos.x0, pos.y0 + pos.height * 0.15, pos.width, pos.height * 0.7])
59 cbar = fig.colorbar(im_dl, cax=cax, orientation='vertical')
60 cbar.set_label("Soil Moisture")
61
62 # Save and show
63 plt.savefig("Obs_vs_CYGNSS_SM_2022.png", dpi=300, bbox_inches='tight')
64 plt.show()

```

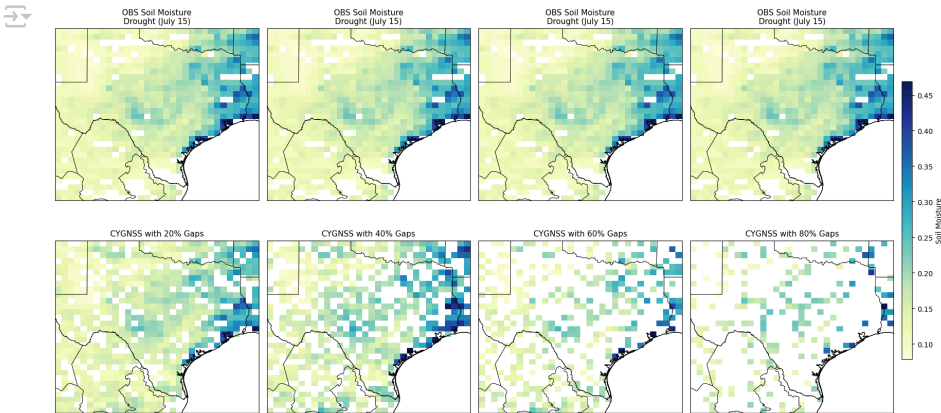


▼ More validation

```

1 import cartopy.crs as ccrs
2 import cartopy.feature as cfeature
3 import matplotlib.pyplot as plt
4 from matplotlib import gridspec
5 import numpy as np
6
7 # Required for plotting in Colab
8 %matplotlib inline
9
10 # Define a single event
11 event_date = "2022-07-15"
12 event_title = "Drought (July 15)"
13 gap_levels = [0.2, 0.4, 0.6, 0.8]
14
15 # Create 2-row grid: Row 1 = Original OBS, Row 2 = CYGNSS with different gaps
16 fig = plt.figure(figsize=(5 * len(gap_levels) + 1.5, 10))
17 gs = gridspec.GridSpec(2, len(gap_levels) + 1, width_ratios=[1]*len(gap_levels) + [0.05], hspace=0.15, wspace=0.05)
18
19 axs_obs = []
20 axs_dl = []
21
22 # Row 1: Plot same OBS data for all columns
23 for i, gap in enumerate(gap_levels):
24     ax = fig.add_subplot(gs[0, i], projection=ccrs.PlateCarree())
25     axs_obs.append(ax)
26
27     obs_data = obs.sel(time=event_date)
28     im_obs = obs_data.plot(ax=ax, transform=ccrs.PlateCarree(), cmap="YlGnBu", add_colorbar=False)
29
30     ax.coastlines()
31     ax.add_feature(cfeature.BORDERS, linestyle=':')
32     ax.add_feature(cfeature.STATES, linewidth=0.5)
33     ax.set_title(f"OBS Soil Moisture\n{event_title}", fontsize=11)
34
35 # Row 2: CYGNSS with artificial gaps at different levels
36 for i, gap in enumerate(gap_levels):
37     ax = fig.add_subplot(gs[1, i], projection=ccrs.PlateCarree())
38     axs_dl.append(ax)
39
40     sm_data = ds_cygnss_gap_fill_final.SM_daily_gap_filled.sel(
41         lat=slice(obs.lat.values[0], obs.lat.values[-1]),
42         lon=slice(obs.lon.values[0], obs.lon.values[-1]),
43         time=event_date
44     )
45
46     masked_sm = sm_data.where(mask == 1)
47
48     # Create artificial gaps
49     rng = np.random.default_rng(seed=42 + i) # different seed per gap
50     gap_mask = rng.uniform(size=masked_sm.shape) > gap # keep (1 - gap)
51     masked_sm = masked_sm.where(gap_mask)
52
53     im_dl = masked_sm.plot(ax=ax, transform=ccrs.PlateCarree(), cmap="YlGnBu", add_colorbar=False)
54
55     ax.coastlines()
56     ax.add_feature(cfeature.BORDERS, linestyle=':')
57     ax.add_feature(cfeature.STATES, linewidth=0.5)
58     ax.set_title(f"CYGNSS with {int(gap*100)}% Gaps", fontsize=11)
59
60 # Shared colorbar
61 cax = fig.add_subplot(gs[:, -1])
62 pos = cax.get_position()
63 cax.set_position([pos.x0, pos.y0 + pos.height * 0.15, pos.width, pos.height * 0.7])
64 cbar = fig.colorbar(im_dl, cax=cax, orientation='vertical')
65 cbar.set_label("Soil Moisture")
66
67 # Save and show
68 plt.savefig("CYGNSS_gap_levels_comparison.png", dpi=300, bbox_inches='tight')
69 plt.show()

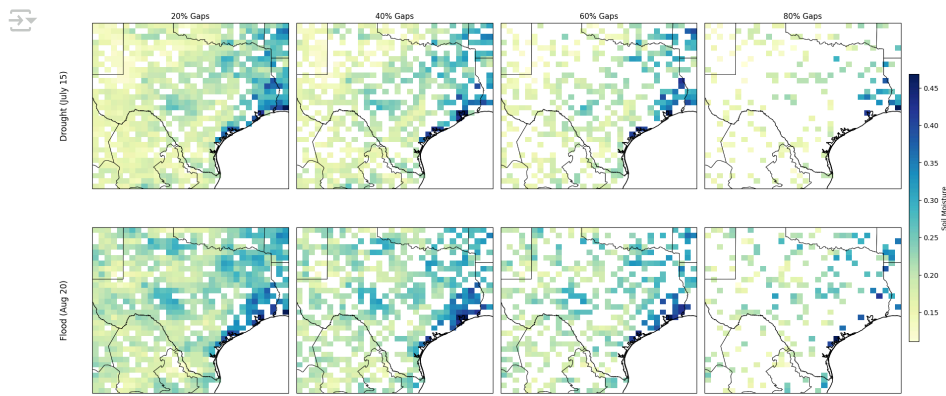
```




```

1 import cartopy.crs as ccrs
2 import cartopy.feature as cfeature
3 import matplotlib.pyplot as plt
4 from matplotlib import gridspec
5 import numpy as np
6
7 # Required for plotting in Colab
8 %matplotlib inline
9
10 # Define events and gap levels
11 event_dict = {
12     "2022-07-15": "Drought (July 15)",
13     "2022-08-20": "Flood (Aug 20)"
14 }
15 gap_levels = [0.2, 0.4, 0.6, 0.8]
16
17 # Create figure grid: 2 rows (events), 4 cols (gap levels), plus 1 for colorbar
18 fig = plt.figure(figsize=(5 * len(gap_levels) + 1.5, 10))
19 gs = gridspec.GridSpec(len(event_dict), len(gap_levels) + 1,
20                         width_ratios=[1]*len(gap_levels) + [0.05],
21                         hspace=0.15, wspace=0.05)
22
23 # Iterate over each event
24 for row_idx, (date_str, title) in enumerate(event_dict.items()):
25     for col_idx, gap in enumerate(gap_levels):
26         ax = fig.add_subplot(gs[row_idx, col_idx], projection=ccrs.PlateCarree())
27
28         sm_data = ds_cygnss_gap_fill_final.SM_daily_gap_filled.sel(
29             lat=slice(obs.lat.values[0], obs.lat.values[-1]),
30             lon=slice(obs.lon.values[0], obs.lon.values[-1]),
31             time=date_str
32         )
33
34         masked_sm = sm_data.where(mask == 1)
35
36         # Create artificial gaps
37         rng = np.random.default_rng(seed=100 * row_idx + col_idx) # unique seed
38         gap_mask = rng.uniform(size=masked_sm.shape) > gap # keep (1 - gap)
39         masked_sm = masked_sm.where(gap_mask)
40
41         im = masked_sm.plot(ax=ax, transform=ccrs.PlateCarree(), cmap="YlGnBu",
42                             add_colorbar=False)
43
44         ax.coastlines()
45         ax.add_feature(cfeature.BORDERS, linestyle=':')
46         ax.add_feature(cfeature.STATES, linewidth=0.5)
47
48         if row_idx == 0:
49             ax.set_title(f"{int(gap * 100)}% Gaps", fontsize=11)
50         else:
51             ax.set_title("", fontsize=11)
52         if col_idx == 0:
53             ax.text(-0.12, 0.5, title, va='center', ha='right',
54                     fontsize=12, rotation=90, transform=ax.transAxes)
55
56 # Shared colorbar
57 cax = fig.add_subplot(gs[:, -1])
58 pos = cax.get_position()
59 cax.set_position([pos.x0, pos.y0 + pos.height * 0.15, pos.width, pos.height * 0.7])
60 cbar = fig.colorbar(im, cax=cax, orientation='vertical')
61 cbar.set_label("Soil Moisture")
62
63 # Save and show
64 plt.savefig("CYGNSS_gap_comparison_drought_flood.png", dpi=300, bbox_inches='tight')
65 plt.show()

```



```
1 masked_sm.shape
```

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

```
1 import xarray as xr
2 import numpy as np
3
4 # Get the original values
5 original = masked_sm
6
7 # Create a new lat coordinate by appending a dummy lat value
8 new_lat = np.append(original.lat.values, original.lat.values[-1] + (original.lat.values[-1] - original.lat.values[-2]))
9
10 # Create an empty (NaN-filled) array with new shape (1, 32, 32)
11 expanded_data = np.full((1, 32, 32), np.nan, dtype=np.float32)
12
13 # Copy existing data into the top 31 rows
14 expanded_data[:, :31, :] = original.values
15
16 # Create a new expanded DataArray
17 masked_sm_expanded = xr.DataArray(
18     expanded_data,
19     dims=["time", "lat", "lon"],
20     coords={
21         "time": original.time,
22         "lat": new_lat,
23         "lon": original.lon
24     }
25 )
26 masked_sm_expanded.shape
```

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

```

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

```

```

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

```

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

```

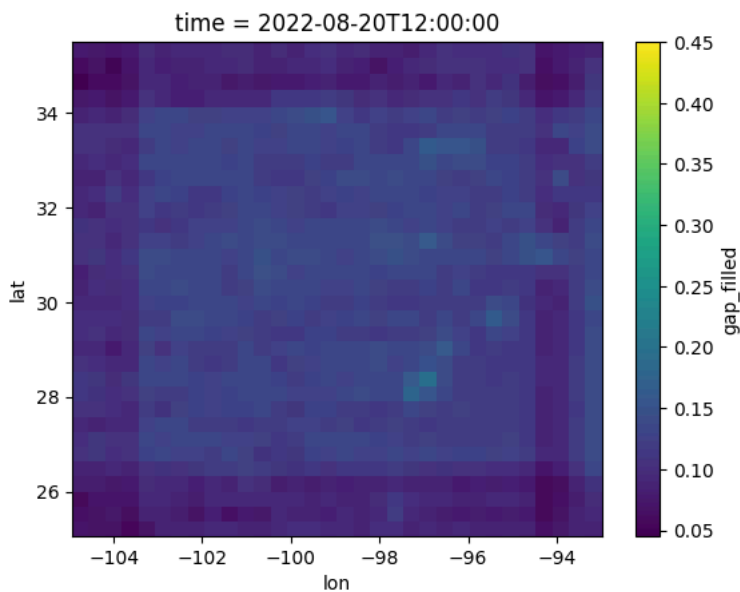
array([[[[0.06985443, 0.06900196, 0.06646743, ..., 0.07475529,
          0.08914833, 0.08713016],
         [0.06458997, 0.06651377, 0.07236394, ..., 0.06712553,
          0.08601389, 0.09326524],
         [0.0569919 , 0.07010572, 0.07185911, ..., 0.06589624,
          0.08138721, 0.09637274],
         ...,
         [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)

```

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

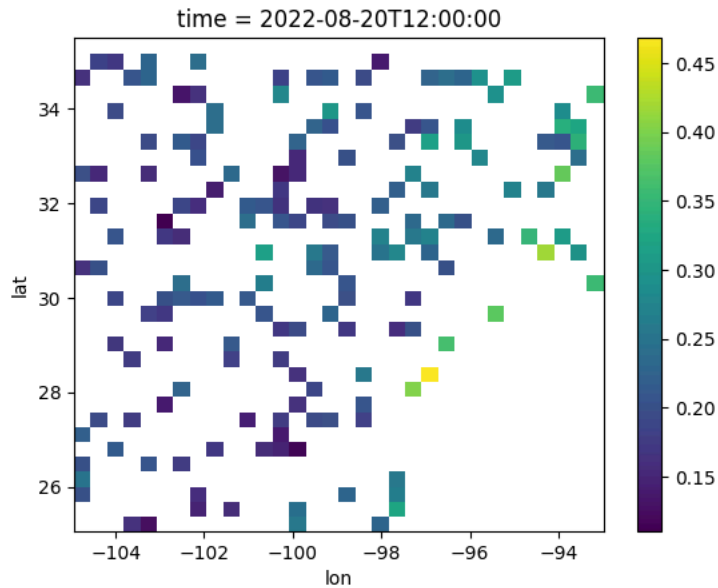
```
1 masked_sm_expanded['gap_filled'].plot(vmax=0.45)
```

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



```
1 masked_sm_expanded.plot()
```


 <matplotlib.collections.QuadMesh at 0x78d95a9e4390>

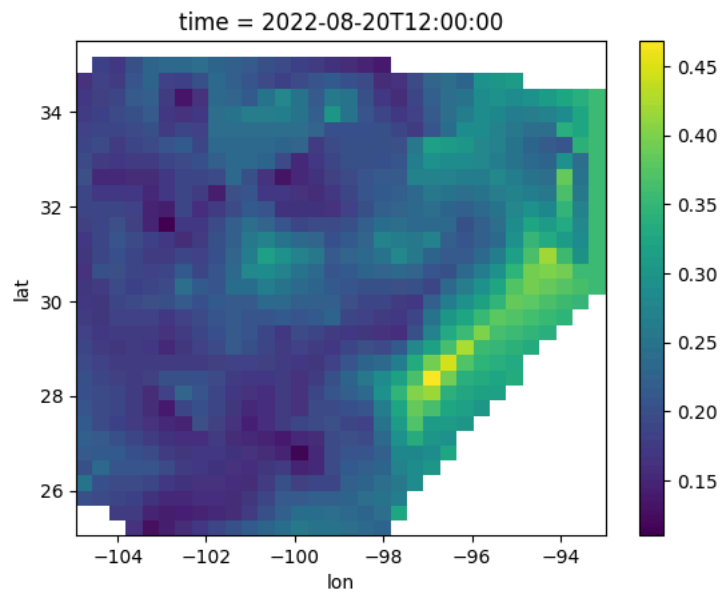


```

1 import numpy as np
2 import xarray as xr
3 from scipy.interpolate import griddata
4
5 def gap_fill_bilinear(data):
6     # Flatten coordinates
7     lon, lat = np.meshgrid(data.lon.values, data.lat.values)
8
9     # Get valid points (non-NaN)
10    valid_mask = ~np.isnan(data.values)
11    points = np.column_stack((lon[valid_mask], lat[valid_mask]))
12    values = data.values[valid_mask]
13
14    # Create grid of all points
15    grid_points = np.column_stack((lon.ravel(), lat.ravel()))
16
17    # Bilinear interpolation (method='linear')
18    filled_values = griddata(points, values, grid_points, method='linear')
19
20    # Reshape to original grid
21    filled_array = filled_values.reshape(data.shape)
22
23    # Replace only NaNs in original with interpolated values
24    filled_combined = np.where(np.isnan(data), filled_array, data)
25
26    # Return new DataArray
27    return xr.DataArray(
28        filled_combined,
29        coords=data.coords,
30        dims=data.dims,
31        name=f"{data.name}_filled" if data.name else None
32    )
33
34
35 1 filled_sm = gap_fill_bilinear(masked_sm_expanded.isel(time=0))
36
37 1 filled_sm.plot()

```

 <matplotlib.collections.QuadMesh at 0x78d97a5adcd0>



```

1 # import numpy as np
2 # import xarray as xr
3 # import matplotlib.pyplot as plt
4 # import cartopy.crs as ccrs
5 # import cartopy.feature as cfeature
6 # from matplotlib import gridspec
7 # import torch
8
9 # # Dummy CYGNSS-like data
10 # lat = np.linspace(25, 35, 32)
11 # lon = np.linspace(-105, -93, 32)
12 # time = np.array(["2022-07-15", "2022-08-20"], dtype="datetime64")
13
14 # # Create random data with NaNs
15 # np.random.seed(0)
16 # data = np.random.rand(2, 32, 32).astype(np.float32)
17 # mask_array = np.random.rand(32, 32) > 0.5
18 # data[:, ~mask_array] = np.nan
19
20 # # Construct dataset and mask
21 # ds_cygnss_gap_fill_final = xr.Dataset({
22 #     "SM_daily_gap_filled": ([ "time", "lat", "lon"], data)
23 # }, coords={"time": time, "lat": lat, "lon": lon})
24
25 # mask = xr.DataArray(mask_array.astype(int), dims=["lat", "lon"], coords={"lat": lat, "lon": lon})
26
27
28
29 # # Event and gap settings
30 # event_dict = {
31 #     "2022-07-15": "Drought (July 15)",
32 #     "2022-08-20": "Flood (Aug 20)"
33 # }
34 # gap_levels = [0.2, 0.4, 0.6, 0.8]
35
36 # # Create figure
37 # fig = plt.figure(figsize=(5 * len(gap_levels) + 1.5, 10))
38 # gs = gridspec.GridSpec(len(event_dict), len(gap_levels) + 1,
39 #                         width_ratios=[1]*len(gap_levels) + [0.05],
40 #                         hspace=0.15, wspace=0.05)
41
42 # for row_idx, (date_str, title) in enumerate(event_dict.items()):
43 #     for col_idx, gap in enumerate(gap_levels):
44 #         ax = fig.add_subplot(gs[row_idx, col_idx], projection=ccrs.PlateCarree())
45
46 #         sm_data = ds_cygnss_gap_fill_final.SM_daily_gap_filled.sel(time=date_str)
47 #         masked_sm = sm_data.where(mask == 1)
48
49 #         rng = np.random.default_rng(seed=100 * row_idx + col_idx)
50 #         gap_mask = rng.uniform(size=masked_sm.shape) > gap
51 #         masked_sm = masked_sm.where(gap_mask)
52
53 #         # Prepare input for model
54 #         img = masked_sm.values
55 #         masks = np.array(create_mask(img))[np.newaxis]
56 #         incomplete_data = np.array([impute_data(img)])
57 #         combined_data = np.concatenate((incomplete_data, masks), axis=0)[np.newaxis]
58
59 #         device = next(model.parameters()).device # Detect if model is on GPU or CPU
60 #         input_tensor = torch.from_numpy(combined_data).float().to(device)
61 #         output = model(input_tensor).cpu().detach().numpy()[0, 0]
62
63 #         # # Model prediction
64 #         # output = model(torch.from_numpy(combined_data).float()).cpu().detach().numpy()[0, 0]
65
66 #         # Plot
67 #         lon_grid, lat_grid = np.meshgrid(masked_sm.lon.values, masked_sm.lat.values)
68 #         im = ax.pcolormesh(lon_grid, lat_grid, output, cmap="YlGnBu", shading="auto", transform=ccrs.PlateCarree())
69
70 #         ax.coastlines()
71 #         ax.add_feature(cfeature.BORDERS, linestyle=':')
72 #         ax.add_feature(cfeature.STATES, linewidth=0.5)
73
74 #         if row_idx == 0:
75 #             ax.set_title(f"{int(gap * 100)}% Gaps", fontsize=11)
76 #         if col_idx == 0:
77 #             ax.text(-0.12, 0.5, title, va='center', ha='right',

```

```
78 #             fontsize=12, rotation=90, transform=ax.transAxes)
79
80 # # Colorbar
81 # cax = fig.add_subplot(gs[:, -1])
82 # pos = cax.get_position()
83 # cax.set_position([pos.x0, pos.y0 + pos.height * 0.15, pos.width, pos.height * 0.7])
84 # cbar = fig.colorbar(im, cax=cax, orientation='vertical')
85 # cbar.set_label("Soil Moisture")
86
87 # plt.savefig("CYGNSS_DL_gapfilled_comparison.png", dpi=300, bbox_inches='tight')
88 # plt.show()
```

```

1 import numpy as np
2 import xarray as xr
3 import matplotlib.pyplot as plt
4 import cartopy.crs as ccrs
5 import cartopy.feature as cfeature
6 from matplotlib import gridspec
7 import torch
8
9
10 # Event definitions
11 event_dict = {
12     "2022-07-15": "Drought (July 15)",
13     "2022-08-20": "Flood (Aug 20)"
14 }
15 gap_levels = [0.2, 0.4, 0.6, 0.8]
16
17 # Set up figure
18 fig = plt.figure(figsize=(5 * len(gap_levels) + 1.5, 10))
19 gs = gridspec.GridSpec(len(event_dict), len(gap_levels) + 1,
20                         width_ratios=[1]*len(gap_levels) + [0.05],
21                         hspace=0.15, wspace=0.05)
22
23 # Land overlay feature to mask oceans
24 ocean = cfeature.NaturalEarthFeature('physical', 'ocean', '10m',
25                                     edgecolor='none', facecolor='white', zorder=10)
26
27 # Loop through events and gaps
28 for row_idx, (date_str, title) in enumerate(event_dict.items()):
29     for col_idx, gap in enumerate(gap_levels):
30         ax = fig.add_subplot(gs[row_idx, col_idx], projection=ccrs.PlateCarree())
31
32         # Select and mask CYGNSS data
33         sm_data = ds_cygnss_gap_fill_final.SM_daily_gap_filled.sel(
34             lat=slice(obs.lat.values[0], obs.lat.values[-1]),
35             lon=slice(obs.lon.values[0], obs.lon.values[-1]),
36             time=date_str
37         )
38         masked_sm = sm_data.where(mask == 1)
39
40         # Artificial gaps
41         rng = np.random.default_rng(seed=100 * row_idx + col_idx)
42         gap_mask = rng.uniform(size=masked_sm.shape) > gap
43         masked_sm = masked_sm.where(gap_mask)
44
45         # # Prepare model input
46         # img = masked_sm.values
47         # masks = np.array(create_mask(img))[np.newaxis]
48         # incomplete_data = np.array([impute_data(img)])
49         # combined_data = np.concatenate((incomplete_data, masks), axis=0)[np.newaxis]
50
51         # Expand to 32x32 if needed
52         img = masked_sm.values
53         # Ensure shape is (1, 32, 32)
54         if img.shape[1] != 32:
55             expanded_img = np.full((1, 32, 32), np.nan, dtype=np.float32)
56             expanded_img[:, :img.shape[1], :] = img
57
58         # Also expand lat coordinate accordingly
59         new_lat = np.append(masked_sm.lat.values,
60                             masked_sm.lat.values[-1] + (masked_sm.lat.values[-1] - masked_sm.lat.values[-2]))
61
62         masked_sm = xr.DataArray(
63             expanded_img,
64             dims=["time", "lat", "lon"],
65             coords={
66                 "time": masked_sm.time,
67                 "lat": new_lat,
68                 "lon": masked_sm.lon
69             }
70         )
71         img = masked_sm.values
72
73         # Prepare model input
74         masks = np.array(create_mask(img[0]))[np.newaxis]
75         incomplete_data = np.array([impute_data(img[0])])
76         combined_data = np.concatenate((incomplete_data, masks), axis=0)[np.newaxis]
77

```



```

78     # Run model
79     device = next(model.parameters()).device
80     input_tensor = torch.from_numpy(combined_data).float().to(device)
81     output = model(input_tensor).cpu().detach().numpy()[0, 0]
82
83     # Plot prediction
84     lon_grid, lat_grid = np.meshgrid(masked_sm.lon.values, masked_sm.lat.values)
85     im = ax.pcolormesh(lon_grid, lat_grid, output, cmap="YlGnBu", shading="auto", transform=ccrs.PlateCarree())
86
87     # Map features
88     ax.coastlines()
89     ax.add_feature(cfeature.BORDERS, linestyle=':')
90     ax.add_feature(cfeature.STATES, linewidth=0.5)
91     ax.add_feature(ocean) # visually mask ocean
92
93     if row_idx == 0:
94         ax.set_title(f"{int(gap * 100)}% Gaps", fontsize=11)
95     if col_idx == 0:
96         ax.text(-0.12, 0.5, title, va='center', ha='right',
97               fontsize=12, rotation=90, transform=ax.transAxes)
98
99 # Colorbar
100 cax = fig.add_subplot(gs[:, -1])
101 pos = cax.get_position()
102 cax.set_position([pos.x0, pos.y0 + pos.height * 0.15, pos.width, pos.height * 0.7])
103 cbar = fig.colorbar(im, cax=cax, orientation='vertical')
104 cbar.set_label("Soil Moisture")
105
106 plt.savefig("CYGNSS_DL_gapfilled_comparison.png", dpi=300, bbox_inches='tight')
107 plt.show()

```

