

Detect where the location of the shipping is where shipping and oil moves are happening in Iran.

```
#Run google earth engine.
import ee
import geemap
import numpy as np
import matplotlib.pyplot as plt

# Earth Engine init
PROJECT_ID = "powerful-tree-490513-v3"
try:
    ee.Initialize(project=PROJECT_ID)
    print("Earth Engine initialized.")
except Exception:
    ee.Authenticate()
    ee.Initialize(project=PROJECT_ID)
    print("Earth Engine authenticated and initialized.")

# Region
region = ee.Geometry.Point([56.25, 26.55])
small_region = region.buffer(40000).bounds()

# Sentinel-1 collection
s1 = (
    ee.ImageCollection("COPERNICUS/S1_GRD")
    .filterBounds(small_region)
    .filterDate("2026-04-01", "2026-05-01")
    .filter(ee.Filter.eq("instrumentMode", "IW"))
    .filter(ee.Filter.listContains("transmitterReceiverPolarisation", "VV"))
    .select("VV")
)

# Coverage filter – only keep images that fully cover our box
region_area = small_region.area(maxError=1)

def add_coverage(img):
    inter_area = img.geometry().intersection(small_region, maxError=1).area(maxError=1)
    return img.set("coverage", inter_area.divide(region_area))

s1 = s1.map(add_coverage).filter(ee.Filter.gte("coverage", 0.95))

n = s1.size().getInfo()
print(f"Fully-covered images in April 2026: {n}")

# Pick the first available fully-covered image
single_image = s1.sort("system:time_start").first()

date = ee.Date(single_image.get("system:time_start")).format("YYYY-MM-dd").getInfo()
print(f"Selected image: {date}")

# To NumPy
```

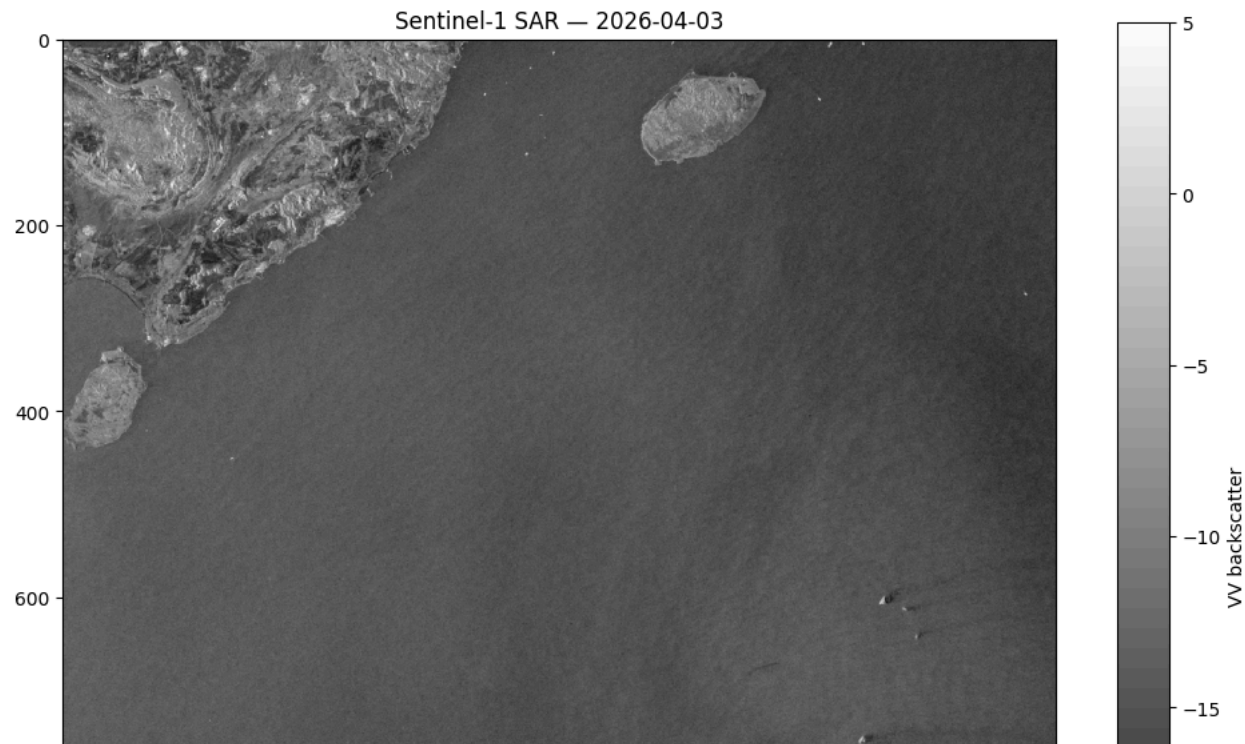
```
# ee_to_numpy
sar_array = geemap.ee_to_numpy(single_image, region=small_region, scale=75)
sar = sar_array[:, :, 0]
print("SAR shape:", sar.shape)

# Plot raw image
plt.figure(figsize=(12, 10))
plt.imshow(sar, cmap="gray", vmin=-25, vmax=5)
plt.title(f"Sentinel-1 SAR - {date}")
plt.colorbar(label="VV backscatter")
plt.show()

# Threshold + ship candidates
threshold = np.nanpercentile(sar, 99.8)
print("Threshold:", threshold)
ship_candidates = sar > threshold

plt.figure(figsize=(12, 10))
plt.imshow(ship_candidates, cmap="gray")
plt.title("Possible Ship Candidates")
plt.show()
```


Earth Engine initialized.
Fully-covered images in April 2026: 10
Selected image: 2026-04-03
SAR shape: (1070, 1069)



```
import ee
import geemap
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import ndimage

# Earth Engine init
PROJECT_ID = "powerful-tree-490513-v3"
try:
    ee.Initialize(project=PROJECT_ID)
    print("Earth Engine initialized.")
except Exception:
    ee.Authenticate()
    ee.Initialize(project=PROJECT_ID)
    print("Earth Engine authenticated and initialized.")
region = ee.Geometry.Point([56.25, 26.55]) #Defining Geographical location where there are most ships occuring.
small_region = region.buffer(40000).bounds()
START_DATE = "2026-04-01"
END_DATE = "2026-05-13"
s1 = (
    ee.ImageCollection("COPERNICUS/S1_GRD")
    .filterBounds(small_region)
    .filterDate(START_DATE, END_DATE)
```

```

        .filter(ee.Filter.eq("instrumentMode", "IW"))
        .filter(ee.Filter.listContains("transmitterReceiverPolarisation", "VV"))
        .select("VV")
    )
    region_area = small_region.area(maxError=1)
    def add_coverage(img):
        inter_area = img.geometry().intersection(small_region, maxError=1).area(maxError=1)
        return img.set("coverage", inter_area.divide(region_area))
    s1 = s1.map(add_coverage).filter(ee.Filter.gte("coverage", 0.95))

    n = s1.size().getInfo()
    print(f"Fully-covered images between {START_DATE} and {END_DATE}: {n}")

    if n == 0:
        raise RuntimeError("No images found. Try widening the date range.")
    image_list = s1.sort("system:time_start").toList(n)

    # -----
    # 5. Loop, download each, count ships
    # -----
    results = []

    for i in range(n):
        img = ee.Image(image_list.get(i))
        date = ee.Date(img.get("system:time_start")).format("YYYY-MM-dd").getInfo()
        print(f"[{i+1}/{n}] Processing {date}...")

        try:
            arr = geemap.ee_to_numpy(img, region=small_region, scale=75)
            sar = arr[:, :, 0].astype(float)

            # Threshold: 99.8th percentile
            threshold = np.nanpercentile(sar, 99.8)
            bright = sar > threshold

            # Connected components, size-filtered
            labeled, num_obj = ndimage.label(bright)
            sizes = ndimage.sum(bright, labeled, range(1, num_obj + 1))
            # Keep components that look ship-sized: 3 to 300 pixels at scale=75
            keep = (sizes >= 3) & (sizes <= 300)
            ship_count = int(keep.sum())

            results.append({
                "date": date,
                "ship_count": ship_count,
                "threshold_db": float(threshold),
            })
            print(f"    → {ship_count} ship candidates (threshold {threshold:.2f} dB)")

        except Exception as e:
            print(f"    → ERROR: {e}")
            results.append({
                "date": date,
                "ship_count": None,
                "threshold_db": None,
            })

```

```
# -----  
# 6. Build dataframe + plot time series  
# -----  
df = pd.DataFrame(results)  
df["date"] = pd.to_datetime(df["date"])  
df = df.sort_values("date").reset_index(drop=True)  
  
print("\n=== Ship count time series ===")  
print(df)  
  
# Save for later  
df.to_csv("hormuz_ship_counts.csv", index=False)  
print("\nSaved: hormuz_ship_counts.csv")  
  
# Plot  
fig, ax = plt.subplots(figsize=(12, 5))  
ax.plot(df["date"], df["ship_count"], marker="o", linewidth=2)  
ax.set_title(f"Hormuz ship candidate count over time ({START_DATE} to {END_DATE})")  
ax.set_xlabel("Date")  
ax.set_ylabel("Ship candidates per image")  
ax.grid(True, alpha=0.3)  
plt.xticks(rotation=45)  
plt.tight_layout()  
plt.show()
```



```

Earth Engine initialized.
Fully-covered images between 2026-04-01 and 2026-05-13: 15
[1/15] Processing 2026-04-03...
  → 158 ship candidates (threshold 3.15 dB)
[2/15] Processing 2026-04-05...
  → 153 ship candidates (threshold 4.01 dB)
[3/15] Processing 2026-04-09...
  → 165 ship candidates (threshold 3.08 dB)
[4/15] Processing 2026-04-11...
  → 149 ship candidates (threshold 4.40 dB)
[5/15] Processing 2026-04-15...
  → 163 ship candidates (threshold 3.27 dB)
[6/15] Processing 2026-04-17...
  → 162 ship candidates (threshold 4.13 dB)
[7/15] Processing 2026-04-21...
  → 163 ship candidates (threshold 3.12 dB)
[8/15] Processing 2026-04-23...

```

```

# STAGE 1 – Real SAR → Flow Network Images
# Pull SAR time series, detect ships per region,
# render each date as a flow-network image.
import ee
import geemap
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.patches import FancyArrowPatch
from scipy import ndimage
from matplotlib.path import Path
from io import BytesIO
from PIL import Image
import os

PROJECT_ID = "powerful-tree-490513-v3"
try:
    ee.Initialize(project=PROJECT_ID)
except Exception:
    ee.Authenticate()
    ee.Initialize(project=PROJECT_ID)

OUT_DIR = "flow_images_real"
os.makedirs(OUT_DIR, exist_ok=True)
# AOI + sub-regions
center = ee.Geometry.Point([56.25, 26.40])
small_region = center.buffer(45000).bounds()
bounds = small_region.coordinates().getInfo()[0]
lons = [p[0] for p in bounds]
lats = [p[1] for p in bounds]
lon_min, lon_max = min(lons), max(lons)
lat_min, lat_max = min(lats), max(lats)

#Plan: divide by the region and apply the regionality into the network flows.
regions_latlon = {
    "North Lane": [(55.95, 26.75), (56.55, 26.75), (56.55, 26.65), (55.95, 26.65)],
    "South Lane": [(55.95, 26.45), (56.55, 26.45), (56.55, 26.05), (55.95, 26.05)],
    "Khasab Bay": [(56.25, 26.65), (56.55, 26.65), (56.55, 26.45), (56.25, 26.45)],
    "Open Gulf West": [(55.95, 26.65), (56.25, 26.65), (56.25, 26.45), (55.95, 26.45)],
}

NODE_NAMES = list(regions_latlon.keys())

```

```

def polygon_centroid(poly):
    return (np.mean([p[0] for p in poly]), np.mean([p[1] for p in poly]))

# Node positions for the flow image, normalized [0,1]
NODE_POSITIONS = np.array([
    [
        (polygon_centroid(regions_latlon[name])[0] - lon_min) / (lon_max - lon_min),
        1 - (polygon_centroid(regions_latlon[name])[1] - lat_min) / (lat_max - lat_min),
    ]
    for name in NODE_NAMES
])

NODE_COLORS = ["red", "blue", "orange", "green"]
IMG_SIZE = 128

```

```

def render_flow_image(positions, counts, img_size=IMG_SIZE, max_count=None):
    n = len(positions)
    fig, ax = plt.subplots(figsize=(img_size/100, img_size/100), dpi=100)
    ax.set_xlim(0, 1); ax.set_ylim(0, 1)
    ax.set_aspect("equal"); ax.axis("off")
    fig.patch.set_facecolor("white")

    if max_count is None:
        max_count = max(max(counts), 1)

    # Thin connecting lines between nodes (visual context)
    for i in range(n):
        for j in range(i+1, n):
            ax.plot([positions[i,0], positions[j,0]],
                    [positions[i,1], positions[j,1]],
                    color="lightgray", linewidth=0.6, zorder=1)

    # Nodes sized by ship count
    for k in range(n):
        c = counts[k]
        size = 30 + 400 * (c / max_count)
        ax.scatter(positions[k,0], positions[k,1],
                   s=size, c=NODE_COLORS[k],
                   alpha=0.8, zorder=3,
                   edgecolors="black", linewidths=0.8)

    buf = BytesIO()
    fig.savefig(buf, format="png", bbox_inches="tight",
               pad_inches=0, dpi=100)
    plt.close(fig)
    buf.seek(0)
    return np.array(Image.open(buf).convert("RGB").resize((img_size, img_size)))

```

```

START_DATE = "2026-04-01"
END_DATE   = "2026-05-13"

```

```

s1 = (
    ee.ImageCollection("COPERNICUS/S1_GRD")
    .filterBounds(small_region)
    .filterDate(START_DATE, END_DATE)
    .filter(ee.Filter.eq("instrumentMode", "IW"))
)

```

```

        .filter(ee.Filter.listContains("transmitterReceiverPolarisation", "VV"))
        .select("VV")
    )

region_area = small_region.area(maxError=1)
def add_coverage(img):
    inter = img.geometry().intersection(small_region, maxError=1).area(maxError=1)
    return img.set("coverage", inter.divide(region_area))

s1 = s1.map(add_coverage).filter(ee.Filter.gte("coverage", 0.95))
n = s1.size().getInfo()
print(f"Fully-covered images: {n}")
if n == 0:
    raise RuntimeError("No images found. Widen date range.")

image_list = s1.sort("system:time_start").toList(n)

#after detecting the ships, we use flow network.
def make_region_masks(H, W):
    def lonlat_to_px(lon, lat):
        x = (lon - lon_min) / (lon_max - lon_min) * W
        y = (lat_max - lat) / (lat_max - lat_min) * H
        return x, y
    yy, xx = np.mgrid[0:H, 0:W]
    pts = np.column_stack([xx.ravel(), yy.ravel()])
    masks = {}
    for name, poly in regions_latlon.items():
        poly_px = [lonlat_to_px(lon, lat) for (lon, lat) in poly]
        masks[name] = Path(poly_px).contains_points(pts).reshape(H, W)
    return masks

records = []
region_masks = None
flow_images = []

for i in range(n):
    img = ee.Image(image_list.get(i))
    date = ee.Date(img.get("system:time_start")).format("YYYY-MM-dd").getInfo()
    print(f"[{i+1}/{n}] {date}...")
    try:
        arr = geemap.ee_to_numpy(img, region=small_region, scale=75)
        sar = arr[:, :, 0].astype(float)
        H, W = sar.shape
        if region_masks is None:
            region_masks = make_region_masks(H, W)

        threshold = np.nanpercentile(sar, 99.8)
        bright = sar > threshold
        labeled, num_obj = ndimage.label(bright)
        sizes = ndimage.sum(bright, labeled, range(1, num_obj + 1))
        keep_ids = np.where((sizes >= 3) & (sizes <= 300))[0] + 1
        centroids = ndimage.center_of_mass(np.isin(labeled, keep_ids),
                                           labeled, keep_ids)

        counts = [0]*len(NODE_NAMES)
        for (cy, cx) in centroids:
            for k, name in enumerate(NODE_NAMES):

```

```

        if region_masks[name][int(cy), int(cx)]:
            counts[k] += 1
            break

    flow_img = render_flow_image(NODE_POSITIONS, counts)
    flow_images.append(flow_img)

    out_path = os.path.join(OUT_DIR, f"flow_{date}.png")
    Image.fromarray(flow_img).save(out_path)

    row = {"date": date, "total": sum(counts)}
    for k, name in enumerate(NODE_NAMES):
        row[name] = counts[k]
    records.append(row)
    print(f"    counts: {dict(zip(NODE_NAMES, counts))}")
except Exception as e:
    print(f"    ERROR: {e}")

df = pd.DataFrame(records)
df["date"] = pd.to_datetime(df["date"])
df = df.sort_values("date").reset_index(drop=True)
flow_images_arr = np.stack(flow_images)
np.save("real_flow_images.npy", flow_images_arr)

print(f"\nSaved {len(flow_images)} flow images to {OUT_DIR}/")
print(f"Saved counts dataframe: real_flow_data.csv")
print(f"Saved image array: real_flow_images.npy  shape={flow_images_arr.shape}")
print(df)

# -----
# Display all flow images as a grid
# -----
nrow = (len(flow_images) + 3) // 4
fig, axes = plt.subplots(nrow, 4, figsize=(14, 3.5*nrow))
axes = np.array(axes).reshape(-1)
for k in range(len(flow_images)):
    axes[k].imshow(flow_images[k])
    axes[k].set_title(df.iloc[k]["date"].strftime("%Y-%m-%d") +
                     f"\ntotal: {df.iloc[k]['total']}", fontsize=10)
    axes[k].axis("off")
for k in range(len(flow_images), len(axes)):
    axes[k].axis("off")
plt.suptitle("Real SAR-derived flow network images (1 per acquisition date)")
plt.tight_layout()
plt.show()

```


Fully-covered images: 7

[1/7] 2026-04-05...

counts: {'North Lane': 0, 'South Lane': 183, 'Khasab Bay': 0, 'Open Gulf West': 0}

```
# STAGE 1 - Real SAR → Flow Network Images
# Pull SAR time series, detect ships per region,
# render each date as a flow-network image.
import ee
import geemap
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.patches import FancyArrowPatch
from scipy import ndimage
from matplotlib.path import Path
from io import BytesIO
from PIL import Image
import os

PROJECT_ID = "powerful-tree-490513-v3"
try:
    ee.Initialize(project=PROJECT_ID)
except Exception:
    ee.Authenticate()
    ee.Initialize(project=PROJECT_ID)

OUT_DIR = "flow_images_real"
os.makedirs(OUT_DIR, exist_ok=True)

# -----
# AOI + sub-regions
# -----
center = ee.Geometry.Point([56.25, 26.40])
small_region = center.buffer(45000).bounds()

bounds = small_region.coordinates().getInfo()[0]
lons = [p[0] for p in bounds]
lats = [p[1] for p in bounds]
lon_min, lon_max = min(lons), max(lons)
lat_min, lat_max = min(lats), max(lats)

regions_latlon = {
    "North Lane": [(55.95, 26.75), (56.55, 26.75), (56.55, 26.65), (55.95, 26.65)],
    "South Lane": [(55.95, 26.45), (56.55, 26.45), (56.55, 26.05), (55.95, 26.05)],
    "Khasab Bay": [(56.25, 26.65), (56.55, 26.65), (56.55, 26.45), (56.25, 26.45)],
    "Open Gulf West": [(55.95, 26.65), (56.25, 26.65), (56.25, 26.45), (55.95, 26.45)],
}

NODE_NAMES = list(regions_latlon.keys())

def polygon_centroid(poly):
    return (np.mean([p[0] for p in poly]), np.mean([p[1] for p in poly]))

# Node positions for the flow image, normalized [0,1]
NODE_POSITIONS = np.array([
    [
        (polygon_centroid(regions_latlon[name])[0] - lon_min) / (lon_max - lon_min),
        1 - (polygon_centroid(regions_latlon[name])[1] - lat_min) / (lat_max - lat_min)
    ]
])
```

```

    ]
    for name in NODE_NAMES
])

NODE_COLORS = ["red", "blue", "orange", "green"]
IMG_SIZE = 128

# -----
# Renderer (shared with Stage 2 – keep style consistent!)
# -----
def render_flow_image(positions, counts, img_size=IMG_SIZE, max_count=None):
    n = len(positions)
    fig, ax = plt.subplots(figsize=(img_size/100, img_size/100), dpi=100)
    ax.set_xlim(0, 1); ax.set_ylim(0, 1)
    ax.set_aspect("equal"); ax.axis("off")
    fig.patch.set_facecolor("white")

    if max_count is None:
        max_count = max(max(counts), 1)

    # Thin connecting lines between nodes (visual context)
    for i in range(n):
        for j in range(i+1, n):
            ax.plot([positions[i,0], positions[j,0]],
                    [positions[i,1], positions[j,1]],
                    color="lightgray", linewidth=0.6, zorder=1)

    # Nodes sized by ship count
    for k in range(n):
        c = counts[k]
        size = 30 + 400 * (c / max_count)
        ax.scatter(positions[k,0], positions[k,1],
                   s=size, c=NODE_COLORS[k],
                   alpha=0.8, zorder=3,
                   edgecolors="black", linewidths=0.8)

    buf = BytesIO()
    fig.savefig(buf, format="png", bbox_inches="tight",
               pad_inches=0, dpi=100)
    plt.close(fig)
    buf.seek(0)
    return np.array(Image.open(buf).convert("RGB").resize((img_size, img_size)))

# -----
# SAR collection
# -----
START_DATE = "2026-04-01"
END_DATE   = "2026-05-13"

s1 = (
    ee.ImageCollection("COPERNICUS/S1_GRD")
    .filterBounds(small_region)
    .filterDate(START_DATE, END_DATE)
    .filter(ee.Filter.eq("instrumentMode", "IW"))
    .filter(ee.Filter.listContains("transmitterReceiverPolarisation", "VV"))
    .select("VV")

```

```

)

region_area = small_region.area(maxError=1)
def add_coverage(img):
    inter = img.geometry().intersection(small_region, maxError=1).area(maxError=1)
    return img.set("coverage", inter.divide(region_area))

s1 = s1.map(add_coverage).filter(ee.Filter.gte("coverage", 0.95))
n = s1.size().getInfo()
print(f"Fully-covered images: {n}")
if n == 0:
    raise RuntimeError("No images found. Widen date range.")

image_list = s1.sort("system:time_start").toList(n)

# -----
# Loop: download, detect ships, count per region, render flow image
# -----
def make_region_masks(H, W):
    def lonlat_to_px(lon, lat):
        x = (lon - lon_min) / (lon_max - lon_min) * W
        y = (lat_max - lat) / (lat_max - lat_min) * H
        return x, y
    yy, xx = np.mgrid[0:H, 0:W]
    pts = np.column_stack([xx.ravel(), yy.ravel()])
    masks = {}
    for name, poly in regions_latlon.items():
        poly_px = [lonlat_to_px(lon, lat) for (lon, lat) in poly]
        masks[name] = Path(poly_px).contains_points(pts).reshape(H, W)
    return masks

records = []
region_masks = None
flow_images = []

for i in range(n):
    img = ee.Image(image_list.get(i))
    date = ee.Date(img.get("system:time_start")).format("YYYY-MM-dd").getInfo()
    print(f"[{i+1}/{n}] {date}...")
    try:
        arr = geemap.ee_to_numpy(img, region=small_region, scale=75)
        sar = arr[:, :, 0].astype(float)
        H, W = sar.shape
        if region_masks is None:
            region_masks = make_region_masks(H, W)

        threshold = np.nanpercentile(sar, 99.8)
        bright = sar > threshold
        labeled, num_obj = ndimage.label(bright)
        sizes = ndimage.sum(bright, labeled, range(1, num_obj + 1))
        keep_ids = np.where((sizes >= 3) & (sizes <= 300))[0] + 1
        centroids = ndimage.center_of_mass(np.isin(labeled, keep_ids),
                                           labeled, keep_ids)

        counts = [0]*len(NODE_NAMES)
        for (cy, cx) in centroids:
            for k, name in enumerate(NODE_NAMES):

```

```

        for k, name in enumerate(NODE_NAMES):
            if region_masks[name][int(cy), int(cx)]:
                counts[k] += 1
                break

    flow_img = render_flow_image(NODE_POSITIONS, counts)
    flow_images.append(flow_img)

    out_path = os.path.join(OUT_DIR, f"flow_{date}.png")
    Image.fromarray(flow_img).save(out_path)

    row = {"date": date, "total": sum(counts)}
    for k, name in enumerate(NODE_NAMES):
        row[name] = counts[k]
    records.append(row)
    print(f"    counts: {dict(zip(NODE_NAMES, counts))}")
except Exception as e:
    print(f"    ERROR: {e}")

df = pd.DataFrame(records)
df["date"] = pd.to_datetime(df["date"])
df = df.sort_values("date").reset_index(drop=True)
df.to_csv("real_flow_data.csv", index=False)

flow_images_arr = np.stack(flow_images)
np.save("real_flow_images.npy", flow_images_arr)

print(f"\nSaved {len(flow_images)} flow images to {OUT_DIR}/")
print(f"Saved counts dataframe: real_flow_data.csv")
print(f"Saved image array: real_flow_images.npy  shape={flow_images_arr.shape}")
print(df)

# Display all flow images as a grid
nrow = (len(flow_images) + 3) // 4
fig, axes = plt.subplots(nrow, 4, figsize=(14, 3.5*nrow))
axes = np.array(axes).reshape(-1)
for k in range(len(flow_images)):
    axes[k].imshow(flow_images[k])
    axes[k].set_title(df.iloc[k]["date"].strftime("%Y-%m-%d") +
                     f"\ntotal: {df.iloc[k]['total']}", fontsize=10)
    axes[k].axis("off")
for k in range(len(flow_images), len(axes)):
    axes[k].axis("off")
plt.suptitle("Real SAR-derived flow network images (1 per acquisition date)")
plt.tight_layout()
plt.show()

```



```
Fully-covered images: 7
[1/7] 2026-04-05...
counts: {'North Lane': 0, 'South Lane': 183, 'Khasab Bay': 0, 'Open Gulf West': 0}
[2/7] 2026-04-11...
```

```
# STAGE 2 – CNN classifier for flow-network state
# Train on synthetic flow images in the SAME STYLE as Stage 1.
# Classes: Low / Medium / High traffic.
# Apply trained model to your real Stage 1 flow images.
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import FancyArrowPatch
from io import BytesIO
from PIL import Image
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader
from tqdm import tqdm

NODE_POSITIONS = np.array([
    [0.5, 0.10], # North Lane (you'll want to load these from Stage 1
    [0.5, 0.85], # South Lane if you re-run – for now hard-coded approx)
    [0.85, 0.45], # Khasab Bay
    [0.15, 0.45], # Open Gulf West
])
NODE_COLORS = ["red", "blue", "orange", "green"]
IMG_SIZE = 128

def render_flow_image(positions, counts, img_size=IMG_SIZE, max_count=None):
    n = len(positions)
    fig, ax = plt.subplots(figsize=(img_size/100, img_size/100), dpi=100)
    ax.set_xlim(0, 1); ax.set_ylim(0, 1)
    ax.set_aspect("equal"); ax.axis("off")
    fig.patch.set_facecolor("white")
    if max_count is None:
        max_count = max(max(counts), 1)
    for i in range(n):
        for j in range(i+1, n):
            ax.plot([positions[i,0], positions[j,0]],
                    [positions[i,1], positions[j,1]],
                    color="lightgray", linewidth=0.6, zorder=1)
    for k in range(n):
        c = counts[k]
        size = 30 + 400 * (c / max_count)
        ax.scatter(positions[k,0], positions[k,1],
                   s=size, c=NODE_COLORS[k],
                   alpha=0.8, zorder=3,
                   edgecolors="black", linewidths=0.8)
    buf = BytesIO()
    fig.savefig(buf, format="png", bbox_inches="tight", pad_inches=0, dpi=100)
    plt.close(fig)
    buf.seek(0)
    return np.array(Image.open(buf).convert("RGB").resize((img_size, img_size)))

#Generate synthetic labeled dataset
#State = total ship count, binned into Low/Med/High
N_SYNTH = 1500
rng = np.random.default_rng(0)
```

```

CLASS_NAMES = ["Low", "Medium", "High"]
# Total-count thresholds: tune to match what you see in real data
LOW_MAX = 15
HIGH_MIN = 40

print(f"Generating {N_SYNT} synthetic labeled flow images...")
synth_imgs = np.zeros((N_SYNT, IMG_SIZE, IMG_SIZE, 3), dtype=np.uint8)
synth_labels = np.zeros(N_SYNT, dtype=np.int64)

for k in tqdm(range(N_SYNT)):
    # Sample a target total ship count uniformly in [0, 80]
    total = rng.integers(0, 80)
    # Split randomly across 4 regions
    splits = rng.dirichlet([1,1,1,1])
    counts = (splits * total).round().astype(int).tolist()
    # Render
    synth_imgs[k] = render_flow_image(NODE_POSITIONS, counts, max_count=80)
    # Label
    if total <= LOW_MAX:
        synth_labels[k] = 0 # Low
    elif total >= HIGH_MIN:
        synth_labels[k] = 2 # High
    else:
        synth_labels[k] = 1 # Medium

print(f"Class distribution: " +
      ", ".join(f"{CLASS_NAMES[c]}={(synth_labels==c).sum()}"
                for c in range(3)))

# 2. PyTorch dataloaders
# Normalize images to [0,1] float, channel-first
X = torch.tensor(synth_imgs).float().permute(0,3,1,2) / 255.0
y = torch.tensor(synth_labels)

# Train/val split
perm = torch.randperm(N_SYNT)
split = int(0.85 * N_SYNT)
X_train, y_train = X[perm[:split]], y[perm[:split]]
X_val, y_val = X[perm[split:]], y[perm[split:]]

train_loader = DataLoader(TensorDataset(X_train, y_train), batch_size=32, shuffle=True)
val_loader = DataLoader(TensorDataset(X_val, y_val), batch_size=32)

# 3. Small CNN
class FlowClassifier(nn.Module):
    def __init__(self, n_classes=3):
        super().__init__()
        self.net = nn.Sequential(
            nn.Conv2d(3, 16, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),
            nn.Conv2d(16, 32, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),
            nn.Conv2d(32, 64, 3, padding=1), nn.ReLU(), nn.AdaptiveAvgPool2d(1),
            nn.Flatten(),
            nn.Linear(64, 32), nn.ReLU(),
            nn.Linear(32, n_classes),
        )
    def forward(self, x):

```

```

        return self.net(x)

device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Training on: {device}")

model = FlowClassifier().to(device)
optim = torch.optim.Adam(model.parameters(), lr=1e-3)
loss_fn = nn.CrossEntropyLoss()

# 4. Train
N_EPOCHS = 8
for epoch in range(N_EPOCHS):
    model.train()
    train_loss = 0; train_correct = 0; train_total = 0
    for xb, yb in train_loader:
        xb, yb = xb.to(device), yb.to(device)
        optim.zero_grad()
        out = model(xb)
        loss = loss_fn(out, yb)
        loss.backward()
        optim.step()
        train_loss += loss.item() * len(xb)
        train_correct += (out.argmax(1) == yb).sum().item()
        train_total += len(xb)

    model.eval()
    val_correct = 0; val_total = 0
    with torch.no_grad():
        for xb, yb in val_loader:
            xb, yb = xb.to(device), yb.to(device)
            out = model(xb)
            val_correct += (out.argmax(1) == yb).sum().item()
            val_total += len(xb)
    print(f"Epoch {epoch+1}/{N_EPOCHS}  "
          f"train_loss={train_loss/train_total:.3f}  "
          f"train_acc={train_correct/train_total:.2%}  "
          f"val_acc={val_correct/val_total:.2%}")

# 5. Apply to REAL flow images from Stage 1
real_imgs = np.load("real_flow_images.npy")      # from Stage 1
import pandas as pd
real_df = pd.read_csv("real_flow_data.csv")

X_real = torch.tensor(real_imgs).float().permute(0,3,1,2) / 255.0
model.eval()
with torch.no_grad():
    out = model(X_real.to(device))
    pred = out.argmax(1).cpu().numpy()
    probs = torch.softmax(out, dim=1).cpu().numpy()

# 6. Show predictions next to each real image
fig, axes = plt.subplots(1, len(real_imgs), figsize=(4*len(real_imgs), 4.5))
if len(real_imgs) == 1:
    axes = [axes]
for k, ax in enumerate(axes):
    ax.imshow(real_imgs[k])
    p = pred[k]

```

```

    conf = probs[k][p]
    date_str = pd.to_datetime(real_df.iloc[k]["date"]).strftime("%Y-%m-%d")
    total = int(real_df.iloc[k]["total"])
    ax.set_title(f"{date_str}\ntotal={total}\nCNN: {CLASS_NAMES[p]} ({conf:.0%})",
                fontsize=10)
    ax.axis("off")
plt.suptitle("CNN classification of real flow-network images")
plt.tight_layout()
plt.show()

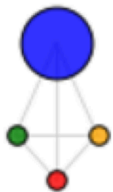
print("\n=== Real-image predictions ===")
for k in range(len(real_imgs)):
    print(f"   {real_df.iloc[k]['date']}: total={int(real_df.iloc[k]['total']):3d} "
          f"→ predicted: {CLASS_NAMES[pred[k]]} (conf={probs[k][pred[k]]:.0%})")

```

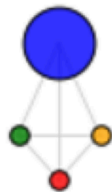
Generating 1500 synthetic labeled flow images...
 100%|██████████| 1500/1500 [00:21<00:00, 71.05it/s]
 Class distribution: Low=314, Medium=441, High=745
 Training on: cpu

Epoch 1/8	train_loss=1.040	train_acc=49.88%	val_acc=48.44%
Epoch 2/8	train_loss=1.032	train_acc=49.88%	val_acc=48.44%
Epoch 3/8	train_loss=1.020	train_acc=49.88%	val_acc=48.44%
Epoch 4/8	train_loss=0.991	train_acc=49.88%	val_acc=48.44%
Epoch 5/8	train_loss=0.952	train_acc=49.88%	val_acc=48.44%
Epoch 6/8	train_loss=0.850	train_acc=54.59%	val_acc=71.11%
Epoch 7/8	train_loss=0.613	train_acc=72.39%	val_acc=79.56%
Epoch 8/8	train_loss=0.484	train_acc=79.45%	val_acc=82.22%

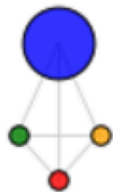
2026-04-05
total=183
CNN: High (99%)



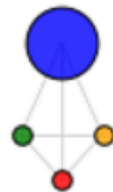
2026-04-11
total=169
CNN: High (99%)



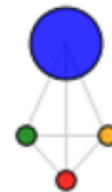
2026-04-17
total=175
CNN: High (99%)



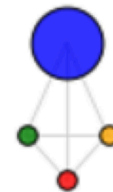
CNN classification of real flow-network images
2026-04-23
total=181
CNN: High (99%)



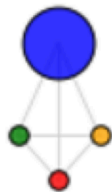
2026-04-29
total=187
CNN: High (99%)



2026-05-05
total=176
CNN: High (99%)



2026-05-11
total=176
CNN: High (99%)



```

=== Real-image predictions ===
2026-04-05: total=183 → predicted: High (conf=99%)
2026-04-11: total=169 → predicted: High (conf=99%)
2026-04-17: total=175 → predicted: High (conf=99%)
2026-04-23: total=181 → predicted: High (conf=99%)
2026-04-29: total=187 → predicted: High (conf=99%)
2026-05-05: total=176 → predicted: High (conf=99%)
2026-05-11: total=176 → predicted: High (conf=99%)

```

```

# STAGE 1 – Real SAR → Flow Network Images
# Pull SAR time series, detect ships per region,
# render each date as a flow-network image.
import ee
import geemap

```

```

import geemap
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import ndimage
from matplotlib.path import Path
from io import BytesIO
from PIL import Image
import os

# Earth Engine init
PROJECT_ID = "powerful-tree-490513-v3"
try:
    ee.Initialize(project=PROJECT_ID)
    print("Earth Engine initialized.")
except Exception:
    ee.Authenticate()
    ee.Initialize(project=PROJECT_ID)

OUT_DIR = "flow_images_real"
os.makedirs(OUT_DIR, exist_ok=True)

center = ee.Geometry.Point([56.25, 26.40])
small_region = center.buffer(45000).bounds()

bounds = small_region.coordinates().getInfo()[0]
lons = [p[0] for p in bounds]; lats = [p[1] for p in bounds]
lon_min, lon_max = min(lons), max(lons)
lat_min, lat_max = min(lats), max(lats)

regions_latlon = {
    "North Lane": [(55.95, 26.75), (56.55, 26.75), (56.55, 26.65), (55.95, 26.65)],
    "South Lane": [(55.95, 26.45), (56.55, 26.45), (56.55, 26.05), (55.95, 26.05)],
    "Khasab Bay": [(56.25, 26.65), (56.55, 26.65), (56.55, 26.45), (56.25, 26.45)],
    "Open Gulf West": [(55.95, 26.65), (56.25, 26.65), (56.25, 26.45), (55.95, 26.45)],
}
NODE_NAMES = list(regions_latlon.keys())
NODE_COLORS = ["red", "blue", "orange", "green"]
IMG_SIZE = 128

def polygon_centroid(poly):
    return (np.mean([p[0] for p in poly]), np.mean([p[1] for p in poly]))

NODE_POSITIONS = np.array([
    [
        (polygon_centroid(regions_latlon[name])[0] - lon_min) / (lon_max - lon_min),
        1 - (polygon_centroid(regions_latlon[name])[1] - lat_min) / (lat_max - lat_min),
    ]
    for name in NODE_NAMES
])

np.save("node_positions.npy", NODE_POSITIONS)

def render_flow_image(positions, counts, img_size=IMG_SIZE, max_count=80):
    n = len(positions)
    fig, ax = plt.subplots(figsize=(img_size/100, img_size/100), dpi=100)
    ax.set_xlim(0, 1); ax.set_ylim(0, 1)

```

```

ax.set_aspect("equal"); ax.axis("off")
fig.patch.set_facecolor("white")
for i in range(n):
    for j in range(i+1, n):
        ax.plot([positions[i,0], positions[j,0]],
                [positions[i,1], positions[j,1]],
                color="lightgray", linewidth=0.6, zorder=1)
for k in range(n):
    c = counts[k]
    size = 30 + 400 * (c / max_count)
    ax.scatter(positions[k,0], positions[k,1],
              s=size, c=NODE_COLORS[k],
              alpha=0.8, zorder=3,
              edgecolors="black", linewidths=0.8)

buf = BytesIO()
fig.savefig(buf, format="png", bbox_inches="tight", pad_inches=0, dpi=100)
plt.close(fig)
buf.seek(0)
return np.array(Image.open(buf).convert("RGB").resize((img_size, img_size)))

```

```
START_DATE = "2026-04-01"
```

```
END_DATE   = "2026-05-13"
```

```

s1 = (
    ee.ImageCollection("COPERNICUS/S1_GRD")
    .filterBounds(small_region)
    .filterDate(START_DATE, END_DATE)
    .filter(ee.Filter.eq("instrumentMode", "IW"))
    .filter(ee.Filter.listContains("transmitterReceiverPolarisation", "VV"))
    .select("VV")
)
region_area = small_region.area(maxError=1)
def add_coverage(img):
    inter = img.geometry().intersection(small_region, maxError=1).area(maxError=1)
    return img.set("coverage", inter.divide(region_area))
s1 = s1.map(add_coverage).filter(ee.Filter.gte("coverage", 0.95))

n = s1.size().getInfo()
print(f"Fully-covered images: {n}")
if n == 0:
    raise RuntimeError("No images. Widen date range or lower coverage threshold to 0.80.")

```

```
image_list = s1.sort("system:time_start").toList(n)
```

```
# Loop: detect ships, count per region, render flow image
```

```

def make_region_masks(H, W):
    def lonlat_to_px(lon, lat):
        x = (lon - lon_min) / (lon_max - lon_min) * W
        y = (lat_max - lat) / (lat_max - lat_min) * H
        return x, y
    yy, xx = np.mgrid[0:H, 0:W]
    pts = np.column_stack([xx.ravel(), yy.ravel()])
    masks = {}
    for name, poly in regions_latlon.items():
        poly_px = [lonlat_to_px(lon, lat) for (lon, lat) in poly]
        masks[name] = Path(poly_px).contains_points(pts).reshape(H, W)
    return masks

```

```

records = []
flow_images = []
region_masks = None

for i in range(n):
    img = ee.Image(image_list.get(i))
    date = ee.Date(img.get("system:time_start")).format("YYYY-MM-dd").getInfo()
    print(f"[{i+1}/{n}] {date}...")
    try:
        arr = geemap.ee_to_numpy(img, region=small_region, scale=75)
        sar = arr[:, :, 0].astype(float)
        H, W = sar.shape
        if region_masks is None:
            region_masks = make_region_masks(H, W)

        threshold = np.nanpercentile(sar, 99.8)
        bright = sar > threshold
        labeled, num_obj = ndimage.label(bright)
        sizes = ndimage.sum(bright, labeled, range(1, num_obj + 1))
        keep_ids = np.where((sizes >= 3) & (sizes <= 300))[0] + 1
        centroids = ndimage.center_of_mass(np.isin(labeled, keep_ids),
                                           labeled, keep_ids)

        counts = [0]*len(NODE_NAMES)
        for (cy, cx) in centroids:
            for k, name in enumerate(NODE_NAMES):
                if region_masks[name][int(cy), int(cx)]:
                    counts[k] += 1
                    break

        flow_img = render_flow_image(NODE_POSITIONS, counts)
        flow_images.append(flow_img)
        Image.fromarray(flow_img).save(os.path.join(OUT_DIR, f"flow_{date}.png"))

        row = {"date": date, "total": sum(counts)}
        for k, name in enumerate(NODE_NAMES):
            row[name] = counts[k]
        records.append(row)
        print(f"    counts: {dict(zip(NODE_NAMES, counts))} total={sum(counts)}")
    except Exception as e:
        print(f"    ERROR: {e}")

df = pd.DataFrame(records)
df["date"] = pd.to_datetime(df["date"])
df = df.sort_values("date").reset_index(drop=True)
df.to_csv("real_flow_data.csv", index=False)
np.save("real_flow_images.npy", np.stack(flow_images))

print(f"\nSaved {len(flow_images)} flow images to {OUT_DIR}/")
print(f"Saved counts: real_flow_data.csv")
print(f"Saved image array: real_flow_images.npy")
print(df)
# Show the grid of real flow images
nrow = (len(flow_images) + 3) // 4
fig, axes = plt.subplots(nrow, 4, figsize=(14, 3.5*nrow))
axes = np.array(axes).reshape(-1)
for k in range(len(flow_images)):

```

```
axes[k].imshow(flow_images[k])
axes[k].set_title(df.iloc[k]["date"].strftime("%Y-%m-%d") +
                  f"\ntotal: {df.iloc[k]['total']}", fontsize=10)
axes[k].axis("off")
for k in range(len(flow_images), len(axes)):
    axes[k].axis("off")
plt.suptitle("Real SAR-derived flow network images")
plt.tight_layout()
plt.show()
```



```
Earth Engine initialized.
Fully-covered images: 7
[1/7] 2026-04-05...
  counts: {'North Lane': 0, 'South Lane': 183, 'Khasab Bay': 0, 'Open Gulf West': 0} total=183
[2/7] 2026-04-11...
  counts: {'North Lane': 0, 'South Lane': 169, 'Khasab Bay': 0, 'Open Gulf West': 0} total=169
```

```
-----
KeyboardInterrupt                                Traceback (most recent call last)
```

```
Cell In[50], line 130
    126 region_masks = None
    127
    128 for i in range(n):
    129     img = ee.Image(image_list.get(i))
--> 130     date = ee.Date(img.get("system:time_start")).format("YYYY-MM-dd").getInfo()
    131     print(f"{i+1}/{n} {date}...")
    132     try:
    133         arr = geemap.ee_to_numpy(img, region=small_region, scale=75)
```

```
File ~/Documents/Risk class project/risk-system-main/.venv/lib/python3.14/site-packages/ee/computedobject.py:108, in ComputedObject.getInfo(self)
```

```
    102 def getInfo(self) -> Any | None:
    103     """Fetch and return information about this object.
    104
    105     Returns:
    106         The object can evaluate to anything.
    107     """
--> 108     return data.computeValue(self)
```

```
File ~/Documents/Risk class project/risk-system-main/.venv/lib/python3.14/site-packages/ee/data.py:1080, in computeValue(obj)
```

```
    1077 body = {'expression': serializer.encode(obj, for_cloud_api=True)}
    1078 _maybe_populate_workload_tag(body)
--> 1080 return _execute_cloud_call(
    1081     _get_cloud_projects()
    1082     .value()
    1083     .compute(body=body, project=_get_projects_path(), prettyPrint=False)
    1084 )['result']
```

```
File ~/Documents/Risk class project/risk-system-main/.venv/lib/python3.14/site-packages/ee/data.py:359, in _execute_cloud_call(call, num_retries)
```

```
    357 num_retries = _get_state().max_retries if num_retries is None else num_retries
    358 try:
--> 359     return call.execute(num_retries=num_retries)
    360 except googleapiclient.errors.HttpError as e:
    361     raise _translate_cloud_exception(e)
```

```
File ~/Documents/Risk class project/risk-system-main/.venv/lib/python3.14/site-packages/googleapiclient/_helpers.py:130, in positional.
```

```
<locals>.positional_decorator.<locals>.positional_wrapper(*args, **kwargs)
    128     elif positional_parameters_enforcement == POSITIONAL_WARNING:
    129         logger.warning(message)
--> 130 return wrapped(*args, **kwargs)
```

```
File ~/Documents/Risk class project/risk-system-main/.venv/lib/python3.14/site-packages/googleapiclient/http.py:923, in HttpRequest.execute(self, http, num_retries)
```

```
    920     self.headers["content-length"] = str(len(self.body))
    922 # Handle retries for server-side errors.
--> 923 resp, content = _retry_request(
    924     http,
    925     num_retries,
    926     "request",
    927     self._sleep,
    928     self._rand,
    929     str(self.uri),
    930     method=str(self.method),
    931     body=self.body,
    932     headers=self.headers,
    933 )
```

```
~~~ ,
935 for callback in self.response_callbacks:
936     callback(resp)
```

File ~/Documents/Risk class project/risk-system-main/.venv/lib/python3.14/site-packages/googleapiclient/http.py:191, in _retry_request(http, num_retries, req_type, sleep, rand, uri, method, *args, **kwargs)

```
189 try:
190     exception = None
--> 191     resp, content = http.request(uri, method, *args, **kwargs)
192 # Retry on SSL errors and socket timeout errors.
193 except _ssl_SLError as ssl_error:
```

File ~/Documents/Risk class project/risk-system-main/.venv/lib/python3.14/site-packages/google_auth_httplib2.py:214, in AuthorizedHttp.request(self, uri, method, body, headers, redirections, connection_type, **kwargs)

```
211     body_stream_position = body.tell()
213 # Make the request.
--> 214 response, content = self.http.request(
215     uri,
216     method,
217     body=body,
218     headers=request_headers,
219     redirections=redirections,
220     connection_type=connection_type,
221     **kwargs
222 )
224 # If the response indicated that the credentials needed to be
225 # refreshed, then refresh the credentials and re-attempt the
226 # request.
227 # A stored token may expire between the time it is retrieved and
228 # the time the request is made, so we may need to try twice.
229 if (
230     response.status in self._refresh_status_codes
231     and _credential_refresh_attempt < self._max_refresh_attempts
232 ):
```

File ~/Documents/Risk class project/risk-system-main/.venv/lib/python3.14/site-packages/ee/_cloud_api_utils.py:79, in _Http.request(**failed resolving arguments)**)

```
72 del redirections # Ignored
74 try:
75     # googleapiclient is expecting an httplib2 object, and doesn't include
76     # requests error in the list of transient errors. Therefore, transient
77     # requests errors should be converted to kinds that googleapiclient
78     # consider transient.
--> 79     response = self._session.request(
80         method, uri, data=body, headers=headers, timeout=self._timeout
81     )
82 except requests.exceptions.ConnectionError as connection_error:
83     raise ConnectionError(connection_error) from connection_error
```

File ~/Documents/Risk class project/risk-system-main/.venv/lib/python3.14/site-packages/requests/sessions.py:592, in Session.request(self, method, url, params, data, headers, cookies, files, auth, timeout, allow_redirects, proxies, hooks, stream, verify, cert, json)

```
587 send_kwargs = {
588     "timeout": timeout,
589     "allow_redirects": allow_redirects,
590 }
591 send_kwargs.update(settings)
--> 592 resp = self.send(prepare, **send_kwargs)
594 return resp
```

File ~/Documents/Risk class project/risk-system-main/.venv/lib/python3.14/site-packages/requests/sessions.py:706, in Session.send(self, request, **kwargs)

```
703 start = preferred_clock()
705 # Send the request
--> 706 r = adapter.send(request, **kwargs)
708 # Total elapsed time of the request (approximately)
709 elapsed = preferred_clock() - start
```

```

File ~/Documents/Risk class project/risk-system-main/.venv/lib/python3.14/site-packages/requests/adapters.py:645, in HTTPAdapter.send(self, request, stream,
timeout, verify, cert, proxies)
    642     timeout = TimeoutSauce(connect=timeout, read=timeout)
    644     try:
--> 645         resp = conn.urlopen(
    646             method=request.method,
    647             url=url,
    648             body=request.body,
    649             headers=request.headers,
    650             redirect=False,
    651             assert_same_host=False,
    652             preload_content=False,
    653             decode_content=False,
    654             retries=self.max_retries,
    655             timeout=timeout,
    656             chunked=chunked,
    657         )
    659     except (ProtocolError, OSError) as err:
    660         raise ConnectionError(err, request=request)

```

```

File ~/Documents/Risk class project/risk-system-main/.venv/lib/python3.14/site-packages/urllib3/connectionpool.py:788, in HTTPConnectionPool.urlopen(self,
method, url, body, headers, retries, redirect, assert_same_host, timeout, pool_timeout, release_conn, chunked, body_pos, preload_content, decode_content,
**response_kw)
    785     response_conn = conn if not release_conn else None
    787     # Make the request on the HTTPConnection object
--> 788     response = self._make_request(
    789         conn,
    790         method,
    791         url,
    792         timeout=timeout_obj,
    793         body=body,
    794         headers=headers,
    795         chunked=chunked,
    796         retries=retries,
    797         response_conn=response_conn,
    798         preload_content=preload_content,
    799         decode_content=decode_content,
    800         **response_kw,
    801     )
    803     # Everything went great!
    804     clean_exit = True

```

```

File ~/Documents/Risk class project/risk-system-main/.venv/lib/python3.14/site-packages/urllib3/connectionpool.py:534, in
HTTPConnectionPool._make_request(self, conn, method, url, body, headers, retries, timeout, chunked, response_conn, preload_content, decode_content,
enforce_content_length)
    532     # Receive the response from the server
    533     try:
--> 534         response = conn.getresponse()
    535     except (BaseSSLError, OSError) as e:
    536         self._raise_timeout(err=e, url=url, timeout_value=read_timeout)

```

```

File ~/Documents/Risk class project/risk-system-main/.venv/lib/python3.14/site-packages/urllib3/connection.py:571, in HTTPConnection.getresponse(self)
    568     _shutdown = getattr(self.sock, "shutdown", None)
    570     # Get the response from http.client.HTTPConnection
--> 571     httplib_response = super().getresponse()
    573     try:
    574         assert_header_parsing(httplib_response.msg)

```

```

File /opt/homebrew/Cellar/python@3.14/3.14.3_1/Frameworks/Python.framework/Versions/3.14/lib/python3.14/http/client.py:1450, in
HTTPConnection.getresponse(self)
    1448     try:
    1449         try:
--> 1450             response.begin()

```

```

# STAGE 2 - CNN classifier (Low / Medium / High activity)
# Train on synthetic flow images in the SAME style as Stage 1.
# Apply trained model to real Stage 1 flow images.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from io import BytesIO
from PIL import Image
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader
from tqdm import tqdm

# Load Stage 1 outputs
NODE_POSITIONS = np.load("node_positions.npy")
real_imgs      = np.load("real_flow_images.npy")
real_df        = pd.read_csv("real_flow_data.csv")

NODE_COLORS = ["red", "blue", "orange", "green"]
IMG_SIZE    = 128

print(f"Loaded {len(real_imgs)} real flow images.")
print(f"Real totals: {list(real_df['total'])}")

# Renderer (MUST match Stage 1)
def render_flow_image(positions, counts, img_size=IMG_SIZE, max_count=80):
    n = len(positions)
    fig, ax = plt.subplots(figsize=(img_size/100, img_size/100), dpi=100)
    ax.set_xlim(0, 1); ax.set_ylim(0, 1)
    ax.set_aspect("equal"); ax.axis("off")
    fig.patch.set_facecolor("white")
    for i in range(n):
        for j in range(i+1, n):
            ax.plot([positions[i,0], positions[j,0]],
                    [positions[i,1], positions[j,1]],
                    color="lightgray", linewidth=0.6, zorder=1)
    for k in range(n):
        c = counts[k]
        size = 30 + 400 * (c / max_count)
        ax.scatter(positions[k,0], positions[k,1],
                   s=size, c=NODE_COLORS[k],
                   alpha=0.8, zorder=3,
                   edgecolors="black", linewidths=0.8)
    buf = BytesIO()
    fig.savefig(buf, format="png", bbox_inches="tight", pad_inches=0, dpi=100)
    plt.close(fig)
    buf.seek(0)
    return np.array(Image.open(buf).convert("RGB").resize((img_size, img_size)))

# Calibrate thresholds from real totals
# (so Low/Med/High labels align with what you actually observe)
real_totals = real_df["total"].values
LOW_MAX     = max(int(np.percentile(real_totals, 33)), 5)
HIGH_MIN    = max(int(np.percentile(real_totals, 67)), LOW_MAX + 1)
print(f"Thresholds: Low <= {LOW_MAX} < Medium < {HIGH_MIN} <= High")

```

```

# 1. Generate synthetic labeled dataset
N_SYNTH = 1500
CLASS_NAMES = ["Low", "Medium", "High"]
rng = np.random.default_rng(0)

# Make synthetic totals span the same range as the real data
max_real = max(real_totals.max(), HIGH_MIN + 10)
print(f"Generating {N_SYNTH} synthetic labeled flow images...")

synth_imgs = np.zeros((N_SYNTH, IMG_SIZE, IMG_SIZE, 3), dtype=np.uint8)
synth_labels = np.zeros(N_SYNTH, dtype=np.int64)

for k in tqdm(range(N_SYNTH)):
    total = int(rng.integers(0, max_real + 5))
    splits = rng.dirichlet([1, 1, 1, 1])
    counts = (splits * total).round().astype(int).tolist()
    synth_imgs[k] = render_flow_image(NODE_POSITIONS, counts, max_count=max_real)
    if total <= LOW_MAX:
        synth_labels[k] = 0
    elif total >= HIGH_MIN:
        synth_labels[k] = 2
    else:
        synth_labels[k] = 1

print("Class counts:",
      ", ".join(f"{CLASS_NAMES[c]}={{(synth_labels==c).sum()}}" for c in range(3)))

# 2. Train / val split, dataloaders
X = torch.tensor(synth_imgs).float().permute(0, 3, 1, 2) / 255.0
y = torch.tensor(synth_labels)

perm = torch.randperm(N_SYNTH)
split = int(0.85 * N_SYNTH)
X_train, y_train = X[perm[:split]], y[perm[:split]]
X_val, y_val = X[perm[split:]], y[perm[split:]]

train_loader = DataLoader(TensorDataset(X_train, y_train), batch_size=32, shuffle=True)
val_loader = DataLoader(TensorDataset(X_val, y_val), batch_size=32)

# 3. Small CNN
class FlowClassifier(nn.Module):
    def __init__(self, n_classes=3):
        super().__init__()
        self.net = nn.Sequential(
            nn.Conv2d(3, 16, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),
            nn.Conv2d(16, 32, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),
            nn.Conv2d(32, 64, 3, padding=1), nn.ReLU(), nn.AdaptiveAvgPool2d(1),
            nn.Flatten(),
            nn.Linear(64, 32), nn.ReLU(),
            nn.Linear(32, n_classes),
        )
    def forward(self, x): return self.net(x)

device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Training on: {device}")

model = FlowClassifier().to(device)
optim = torch.optim.Adam(model.parameters(), lr=1e-3)
loss_fn = nn.CrossEntropyLoss()

```

```
loss_fn = nn.CrossEntropyLoss()
```

```
# 4. Train
```

```
N_EPOCHS = 8
```

```
for epoch in range(N_EPOCHS):
```

```
    model.train()
```

```
    tr_loss = tr_correct = tr_total = 0
```

```
    for xb, yb in train_loader:
```

```
        xb, yb = xb.to(device), yb.to(device)
```

```
        optim.zero_grad()
```

```
        out = model(xb)
```

```
        loss = loss_fn(out, yb)
```

```
        loss.backward()
```

```
        optim.step()
```

```
        tr_loss += loss.item() * len(xb)
```

```
        tr_correct += (out.argmax(1) == yb).sum().item()
```

```
        tr_total += len(xb)
```

```
    model.eval()
```

```
    v_correct = v_total = 0
```

```
    with torch.no_grad():
```

```
        for xb, yb in val_loader:
```

```
            xb, yb = xb.to(device), yb.to(device)
```

```
            out = model(xb)
```

```
            v_correct += (out.argmax(1) == yb).sum().item()
```

```
            v_total += len(xb)
```

```
    print(f"Epoch {epoch+1}/{N_EPOCHS} "
```

```
          f"loss={tr_loss/tr_total:.3f} "
```

```
          f"train_acc={tr_correct/tr_total:.1%} "
```

```
          f"val_acc={v_correct/v_total:.1%}")
```

```
# 5. Apply to real flow images
```

```
X_real = torch.tensor(real_imgs).float().permute(0, 3, 1, 2) / 255.0
```

```
model.eval()
```

```
with torch.no_grad():
```

```
    out = model(X_real.to(device))
```

```
    pred = out.argmax(1).cpu().numpy()
```

```
    probs = torch.softmax(out, dim=1).cpu().numpy()
```

```
# 6. Show predictions next to each real image
```

```
n_real = len(real_imgs)
```

```
fig, axes = plt.subplots(1, n_real, figsize=(4*n_real, 4.5))
```

```
if n_real == 1: axes = [axes]
```

```
for k, ax in enumerate(axes):
```

```
    ax.imshow(real_imgs[k])
```

```
    p = pred[k]; conf = probs[k][p]
```

```
    date_str = pd.to_datetime(real_df.iloc[k]["date"]).strftime("%Y-%m-%d")
```

```
    total = int(real_df.iloc[k]["total"])
```

```
    ax.set_title(f"{date_str}\ntotal={total}\nCNN: {CLASS_NAMES[p]} ({conf:.0%})",  
                fontsize=10)
```

```
    ax.axis("off")
```

```
plt.suptitle("CNN classification of real flow-network images")
```

```
plt.tight_layout()
```

```
plt.show()
```

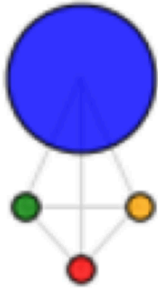
```
print("\n=== Real-image predictions ===")
```

```
for k in range(n_real):
```

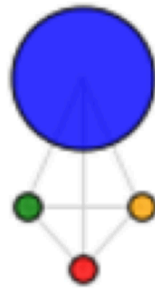
```
print(f" {real_df.iloc[k]['date']}: total={int(real_df.iloc[k]['total']):3d}"
      f" → {CLASS_NAMES[pred[k]]} (conf={probs[k][pred[k]].0%})")
```

Loaded 5 real flow images.
 Real totals: [183, 169, 175, 181, 187]
 Thresholds: Low <= 176 < Medium < 182 <= High
 Generating 1500 synthetic labeled flow images...
 100%|██████████| 1500/1500 [00:18<00:00, 79.13it/s]
 Class counts: Low=1348, Medium=44, High=108
 Training on: cpu
 Epoch 1/8 loss=0.609 train_acc=77.2% val_acc=91.1%
 Epoch 2/8 loss=0.397 train_acc=89.6% val_acc=91.1%
 Epoch 3/8 loss=0.397 train_acc=89.6% val_acc=91.1%
 Epoch 4/8 loss=0.401 train_acc=89.6% val_acc=91.1%
 Epoch 5/8 loss=0.394 train_acc=89.6% val_acc=91.1%
 Epoch 6/8 loss=0.392 train_acc=89.6% val_acc=91.1%
 Epoch 7/8 loss=0.398 train_acc=89.6% val_acc=91.1%
 Epoch 8/8 loss=0.399 train_acc=89.6% val_acc=91.1%

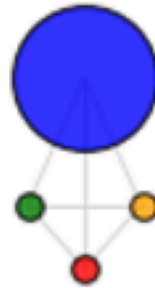
2026-04-05
 total=183
 CNN: Low (82%)



2026-04-11
 total=169
 CNN: Low (83%)



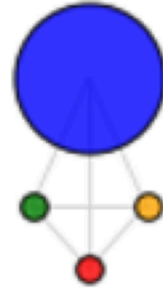
CNN classification of real flow-network images
 2026-04-17
 total=175
 CNN: Low (82%)



2026-04-23
 total=181
 CNN: Low (82%)



2026-04-29
 total=187
 CNN: Low (82%)



=== Real-image predictions ===

2026-04-05: total=183 → Low (conf=82%)
 2026-04-11: total=169 → Low (conf=83%)
 2026-04-17: total=175 → Low (conf=82%)
 2026-04-23: total=181 → Low (conf=82%)
 2026-04-29: total=187 → Low (conf=82%)

Testing with Position based flow images instead to capture the movements of clusters in oil ships.

```
# STAGE 1 – Real SAR → Position-based Flow Images
# Each ship is a dot at its actual location.
# Spatial distribution is what we'll classify.
```

```
import ee
import geemap
```

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import ndimage
from io import BytesIO
from PIL import Image
import os

PROJECT_ID = "powerful-tree-490513-v3"
try:
    ee.Initialize(project=PROJECT_ID)
    print("Earth Engine initialized.")
except Exception:
    ee.Authenticate()
    ee.Initialize(project=PROJECT_ID)

OUT_DIR = "flow_images_real"
os.makedirs(OUT_DIR, exist_ok=True)

IMG_SIZE = 128

# AOI
center = ee.Geometry.Point([56.25, 26.40])
small_region = center.buffer(45000).bounds()

# Renderer: each ship = small dot at its (x,y)
# Positions are normalized to [0, 1] for portability
def render_positions_image(positions_norm, img_size=IMG_SIZE):
    fig, ax = plt.subplots(figsize=(img_size/100, img_size/100), dpi=100)
    ax.set_xlim(0, 1); ax.set_ylim(0, 1)
    ax.set_aspect("equal"); ax.axis("off")
    fig.patch.set_facecolor("white")
    if len(positions_norm) > 0:
        xs = positions_norm[:, 0]; ys = positions_norm[:, 1]
        ax.scatter(xs, ys, s=18, c="steelblue",
                  alpha=0.85, edgecolors="black", linewidths=0.4)
    buf = BytesIO()
    fig.savefig(buf, format="png", bbox_inches="tight", pad_inches=0, dpi=100)
    plt.close(fig)
    buf.seek(0)
    return np.array(Image.open(buf).convert("RGB").resize((img_size, img_size)))

# Pull SAR collection
START_DATE = "2026-04-01"
END_DATE = "2026-05-01"

s1 = (
    ee.ImageCollection("COPERNICUS/S1_GRD")
    .filterBounds(small_region)
    .filterDate(START_DATE, END_DATE)
    .filter(ee.Filter.eq("instrumentMode", "IW"))
    .filter(ee.Filter.listContains("transmitterReceiverPolarisation", "VV"))
    .select("VV")
)

region_area = small_region.area(maxError=1)
def add_coverage(img):
    inter = img.geometry().intersection(small_region, maxError=1).area(maxError=1)

```

```

        return img.set("coverage", inter.divide(region_area))
s1 = s1.map(add_coverage).filter(ee.Filter.gte("coverage", 0.95))

n = s1.size().getInfo()
print(f"Fully-covered images: {n}")
if n == 0:
    raise RuntimeError("No images. Widen date range or lower coverage threshold.")

image_list = s1.sort("system:time_start").toList(n)

# Loop: detect ships, save positions, render
records = []
flow_images = []
all_positions = [] # list of np arrays, one per image

for i in range(n):
    img = ee.Image(image_list.get(i))
    date = ee.Date(img.get("system:time_start")).format("YYYY-MM-dd").getInfo()
    print(f"[{i+1}/{n}] {date}...")
    try:
        arr = geemap.ee_to_numpy(img, region=small_region, scale=75)
        sar = arr[:, :, 0].astype(float)
        H, W = sar.shape

        threshold = np.nanpercentile(sar, 99.8)
        bright = sar > threshold
        labeled, num_obj = ndimage.label(bright)
        sizes = ndimage.sum(bright, labeled, range(1, num_obj + 1))
        keep_ids = np.where((sizes >= 3) & (sizes <= 300))[0] + 1
        centroids = ndimage.center_of_mass(np.isin(labeled, keep_ids),
                                             labeled, keep_ids)

        # Normalize positions to [0, 1]
        positions_norm = np.array(
            [(cx / W, cy / H) for (cy, cx) in centroids]
        ) if len(centroids) > 0 else np.zeros((0, 2))

        flow_img = render_positions_image(positions_norm)
        flow_images.append(flow_img)
        all_positions.append(positions_norm)
        Image.fromarray(flow_img).save(os.path.join(OUT_DIR, f"flow_{date}.png"))

    # Spatial statistics (will help us calibrate Stage 2 thresholds)
    if len(positions_norm) >= 2:
        from scipy.spatial.distance import pdist
        d = pdist(positions_norm)
        mean_nn = float(np.min(
            pdist(positions_norm).reshape(-1)[1:]
        )) if False else None
        # Mean nearest-neighbor distance:
        from scipy.spatial import cKDTree
        tree = cKDTree(positions_norm)
        dists, _ = tree.query(positions_norm, k=2)
        mean_nn = float(dists[:, 1].mean())
        spread = float(positions_norm.std(axis=0).mean())
    else:
        mean_nn = None
        spread = None

```

```

        records.append({
            "date": date,
            "n_ships": len(positions_norm),
            "mean_nn_distance": mean_nn,
            "spread": spread,
        })
    print(f"    ships={len(positions_norm)}, "
          f"mean_nn={mean_nn:.3f if mean_nn else 'NA'}, "
          f"spread={spread:.3f if spread else 'NA'}")

except Exception as e:
    print(f"    ERROR: {e}")

# Save outputs
df = pd.DataFrame(records)
df["date"] = pd.to_datetime(df["date"])
df = df.sort_values("date").reset_index(drop=True)
df.to_csv("real_flow_data.csv", index=False)
np.save("real_flow_images.npy", np.stack(flow_images))

# Save positions list (one array per image)
np.savez("real_positions.npz",
        *[all_positions[i] for i in range(len(all_positions))])

print(f"\nSaved {len(flow_images)} flow images to {OUT_DIR}/")
print("Saved: real_flow_data.csv, real_flow_images.npy, real_positions.npz")
print(df)

# Show grid
nrow = (len(flow_images) + 3) // 4
fig, axes = plt.subplots(nrow, 4, figsize=(14, 3.5*nrow))
axes = np.array(axes).reshape(-1)
for k in range(len(flow_images)):
    axes[k].imshow(flow_images[k])
    axes[k].set_title(
        df.iloc[k]["date"].strftime("%Y-%m-%d") +
        f"\nships: {df.iloc[k]['n_ships']}", fontsize=10)
    axes[k].axis("off")
for k in range(len(flow_images), len(axes)):
    axes[k].axis("off")
plt.suptitle("Real SAR-derived position flow images (each dot = one ship)")
plt.tight_layout()
plt.show()

```

```
Earth Engine initialized.
Fully-covered images: 5
[1/5] 2026-04-05...
  ERROR: Invalid format specifier '.3f if mean_nn else 'NA'' for object of type 'float'
[2/5] 2026-04-11...
  ERROR: Invalid format specifier '.3f if mean_nn else 'NA'' for object of type 'float'
[3/5] 2026-04-17...
  ERROR: Invalid format specifier '.3f if mean_nn else 'NA'' for object of type 'float'
[4/5] 2026-04-23...
  ERROR: Invalid format specifier '.3f if mean_nn else 'NA'' for object of type 'float'
[5/5] 2026-04-29...
  ERROR: Invalid format specifier '.3f if mean_nn else 'NA'' for object of type 'float'
```

Saved 5 flow images to flow_images_real/
Saved: real_flow_data.csv, real_flow_images.npy, real_positions.npz

	date	n_ships	mean_nn_distance	spread
0	2026-04-05	242	0.010673	0.115192
1	2026-04-11	228	0.011631	0.125859
2	2026-04-17	234	0.008865	0.109926
3	2026-04-23	237	0.010363	0.120071
4	2026-04-29	249	0.010422	0.114796

Real SAR-derived position flow images (each dot = one ship)



```

# STAGE 2 – CNN classifier on spatial tightness
# Classes: Tight / Mixed / Spread
# Trained on synthetic position flow images in the same style as Stage 1.
# Applied to real Stage 1 flow images.

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from io import BytesIO
from PIL import Image
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader
from scipy.spatial import cKDTree
from tqdm import tqdm

# Load Stage 1 outputs
real_imgs = np.load("real_flow_images.npy")
real_df = pd.read_csv("real_flow_data.csv")

IMG_SIZE = 128
CLASS_NAMES = ["Tight", "Mixed", "Spread"]
print(f"Loaded {len(real_imgs)} real flow images.")

# Renderer: must match Stage 1 exactly
def render_positions_image(positions_norm, img_size=IMG_SIZE):
    fig, ax = plt.subplots(figsize=(img_size/100, img_size/100), dpi=100)
    ax.set_xlim(0, 1); ax.set_ylim(0, 1)
    ax.set_aspect("equal"); ax.axis("off")
    fig.patch.set_facecolor("white")
    if len(positions_norm) > 0:
        ax.scatter(positions_norm[:,0], positions_norm[:,1],
                    s=18, c="steelblue",
                    alpha=0.85, edgecolors="black", linewidths=0.4)
    buf = BytesIO()
    fig.savefig(buf, format="png", bbox_inches="tight", pad_inches=0, dpi=100)
    plt.close(fig)
    buf.seek(0)
    return np.array(Image.open(buf).convert("RGB").resize((img_size, img_size)))

# Synthetic data generator for tightness classes
def sample_positions_for_class(label, n_ships, rng):
    """
    label = 0 (Tight): all ships clustered in one small area
    label = 1 (Mixed): two clusters
    label = 2 (Spread): random positions across the whole image
    """
    if label == 0: # Tight
        center = rng.uniform(0.2, 0.8, size=2)
        pts = center + rng.normal(0, 0.04, size=(n_ships, 2))
    elif label == 1: # Mixed (two clusters)
        n1 = n_ships // 2
        n2 = n_ships - n1
        c1 = rng.uniform(0.15, 0.4, size=2)
        c2 = rng.uniform(0.6, 0.85, size=2)
        pts = np.vstack([
            c1 + rng.normal(0, 0.05, size=(n1, 2)),
            c2 + rng.normal(0, 0.05, size=(n2, 2)),

```

```

    })
    else: # Spread
        pts = rng.uniform(0.05, 0.95, size=(n_ships, 2))
        return np.clip(pts, 0, 1)

# Generate synthetic training set
N_SYNTH = 1500
rng = np.random.default_rng(0)

# Match number of ships to real range, with variation
real_n = real_df["n_ships"].values
min_ships = max(int(real_n.min()), 5)
max_ships = max(int(real_n.max()), 30)

print(f"Generating {N_SYNTH} synthetic labeled images "
      f"(n_ships in [{min_ships}, {max_ships}])...")

synth_imgs = np.zeros((N_SYNTH, IMG_SIZE, IMG_SIZE, 3), dtype=np.uint8)
synth_labels = np.zeros(N_SYNTH, dtype=np.int64)

for k in tqdm(range(N_SYNTH)):
    label = rng.integers(0, 3)
    n_ships = int(rng.integers(min_ships, max_ships + 1))
    pts = sample_positions_for_class(label, n_ships, rng)
    synth_imgs[k] = render_positions_image(pts)
    synth_labels[k] = label

print("Class counts:",
      ", ".join(f"{CLASS_NAMES[c]}={(synth_labels==c).sum()}" for c in range(3)))

# PyTorch setup
X = torch.tensor(synth_imgs).float().permute(0, 3, 1, 2) / 255.0
y = torch.tensor(synth_labels)

perm = torch.randperm(N_SYNTH)
split = int(0.85 * N_SYNTH)
X_train, y_train = X[perm[:split]], y[perm[:split]]
X_val, y_val = X[perm[split:]], y[perm[split:]]

train_loader = DataLoader(TensorDataset(X_train, y_train), batch_size=32, shuffle=True)
val_loader = DataLoader(TensorDataset(X_val, y_val), batch_size=32)

class FlowClassifier(nn.Module):
    def __init__(self, n_classes=3):
        super().__init__()
        self.net = nn.Sequential(
            nn.Conv2d(3, 16, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),
            nn.Conv2d(16, 32, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),
            nn.Conv2d(32, 64, 3, padding=1), nn.ReLU(), nn.AdaptiveAvgPool2d(1),
            nn.Flatten(),
            nn.Linear(64, 32), nn.ReLU(),
            nn.Linear(32, n_classes),
        )
    def forward(self, x): return self.net(x)

device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Training on: {device}")

```

```

model = FlowClassifier().to(device)
optim = torch.optim.Adam(model.parameters(), lr=1e-3)
loss_fn = nn.CrossEntropyLoss()

# Train
N_EPOCHS = 8
for epoch in range(N_EPOCHS):
    model.train()
    tr_loss = tr_correct = tr_total = 0
    for xb, yb in train_loader:
        xb, yb = xb.to(device), yb.to(device)
        optim.zero_grad()
        out = model(xb)
        loss = loss_fn(out, yb)
        loss.backward()
        optim.step()
        tr_loss += loss.item() * len(xb)
        tr_correct += (out.argmax(1) == yb).sum().item()
        tr_total += len(xb)

    model.eval()
    v_correct = v_total = 0
    with torch.no_grad():
        for xb, yb in val_loader:
            xb, yb = xb.to(device), yb.to(device)
            out = model(xb)
            v_correct += (out.argmax(1) == yb).sum().item()
            v_total += len(xb)
    print(f"Epoch {epoch+1}/{N_EPOCHS}  "
          f"loss={tr_loss/tr_total:.3f}  "
          f"train_acc={tr_correct/tr_total:.1%}  "
          f"val_acc={v_correct/v_total:.1%}")

# Apply to real images
X_real = torch.tensor(real_imgs).float().permute(0, 3, 1, 2) / 255.0
model.eval()
with torch.no_grad():
    out = model(X_real.to(device))
    pred = out.argmax(1).cpu().numpy()
    probs = torch.softmax(out, dim=1).cpu().numpy()

# Show predictions
n_real = len(real_imgs)
fig, axes = plt.subplots(1, n_real, figsize=(4*n_real, 4.5))
if n_real == 1: axes = [axes]
for k, ax in enumerate(axes):
    ax.imshow(real_imgs[k])
    p = pred[k]; conf = probs[k][p]
    date_str = pd.to_datetime(real_df.iloc[k]["date"]).strftime("%Y-%m-%d")
    nn = real_df.iloc[k]["mean_nn_distance"]
    nn_str = f"{nn:.3f}" if pd.notna(nn) else "NA"
    ax.set_title(f"{date_str}\nships={int(real_df.iloc[k]['n_ships'])}\n"
                f"mean_nn={nn_str}\nCNN: {CLASS_NAMES[p]} ({conf:.0%})",
                fontsize=9)
    ax.axis("off")
plt.suptitle("CNN classification: Tight / Mixed / Spread")
plt.tight_layout()

```

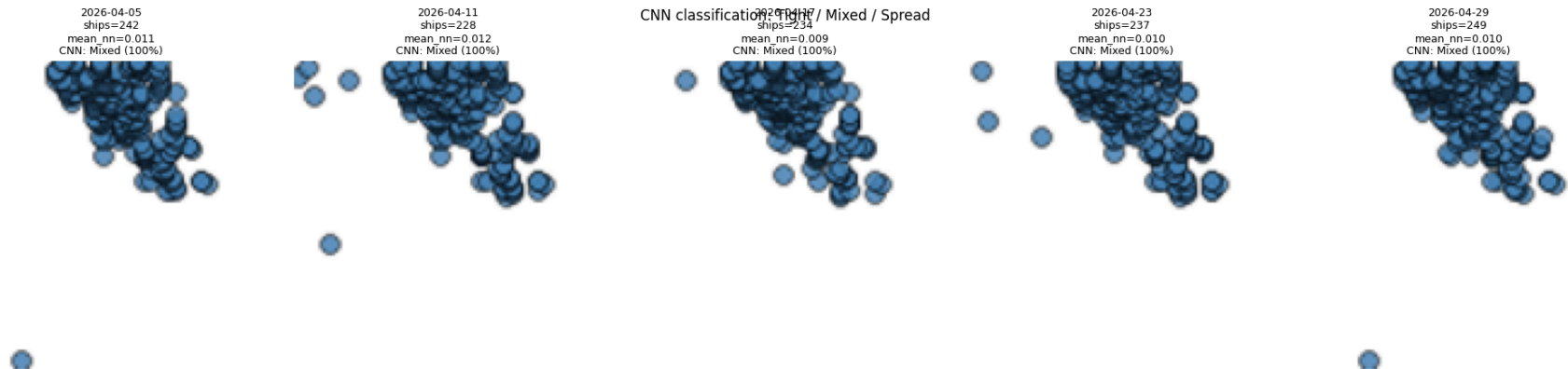
```
plt.show()

print("\n=== Real-image predictions ===")
for k in range(n_real):
    nn = real_df.iloc[k]["mean_nn_distance"]
    print(f"  {real_df.iloc[k]['date']}: "
          f"ships={int(real_df.iloc[k]['n_ships']):3d}  "
          f"mean_nn={nn:.3f if pd.notna(nn) else 'NA'}  "
          f"-> {CLASS_NAMES[pred[k]]} (conf={probs[k][pred[k]]:.0%})")
```

```

Loaded 5 real flow images.
Generating 1500 synthetic labeled images (n_ships in [228, 249])...
100%|██████████| 1500/1500 [00:08<00:00, 169.34it/s]
Class counts: Tight=521, Mixed=494, Spread=485
Training on: cpu
Epoch 1/8  loss=0.946  train_acc=59.9%  val_acc=100.0%
Epoch 2/8  loss=0.338  train_acc=91.1%  val_acc=100.0%
Epoch 3/8  loss=0.083  train_acc=100.0%  val_acc=100.0%
Epoch 4/8  loss=0.013  train_acc=100.0%  val_acc=100.0%
Epoch 5/8  loss=0.004  train_acc=100.0%  val_acc=100.0%
Epoch 6/8  loss=0.003  train_acc=100.0%  val_acc=100.0%
Epoch 7/8  loss=0.002  train_acc=100.0%  val_acc=100.0%
Epoch 8/8  loss=0.001  train_acc=100.0%  val_acc=100.0%

```



=== Real-image predictions ===

```

ValueError                                Traceback (most recent call last)
Cell In[59], line 203
    199 for k in range(n_real):
    200     nn = real_df.iloc[k]["mean_nn_distance"]
    201     print(f" {real_df.iloc[k]['date']}: "
    202           f"ships={int(real_df.iloc[k]['n_ships']):3d} "
--> 203           f"mean_nn={nn:.3f if pd.notna(nn) else 'NA'} "
    204           f"→ {CLASS_NAMES[pred[k]]} (conf={probs[k][pred[k]]:.0%})")

ValueError: Invalid format specifier '.3f if pd.notna(nn) else 'NA'' for object of type 'float'

```

Final Stage: Using the result as prediction model for what is the next images.

```

from PIL import Image
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader
from scipy import ndimage

```

```

from matplotlib.path import Path
from tqdm import tqdm

IMG_SIZE = 128
SEQ_LEN = 3          # use past 3 images to predict the 4th
N_NODES = 4
NODE_COLORS = ["red", "blue", "orange", "green"]

# Same node layout as before (loaded if available)
try:
    NODE_POSITIONS = np.load("node_positions.npy")
except FileNotFoundError:
    # Approximate layout matching Strait of Hormuz regions
    NODE_POSITIONS = np.array([
        [0.5, 0.15], # North Lane
        [0.5, 0.80], # South Lane
        [0.75, 0.45], # Khasab Bay
        [0.25, 0.45], # Open Gulf West
    ])
    np.save("node_positions.npy", NODE_POSITIONS)

# Renderer for the count-based view (4 colored circles)
def render_count_image(counts, img_size=IMG_SIZE, max_count=80):
    n = len(NODE_POSITIONS)
    fig, ax = plt.subplots(figsize=(img_size/100, img_size/100), dpi=100)
    ax.set_xlim(0, 1); ax.set_ylim(0, 1)
    ax.set_aspect("equal"); ax.axis("off")
    fig.patch.set_facecolor("white")
    for i in range(n):
        for j in range(i+1, n):
            ax.plot([NODE_POSITIONS[i,0], NODE_POSITIONS[j,0]],
                    [NODE_POSITIONS[i,1], NODE_POSITIONS[j,1]],
                    color="lightgray", linewidth=0.6, zorder=1)
    for k in range(n):
        c = max(counts[k], 0)
        size = 30 + 400 * (c / max_count)
        ax.scatter(NODE_POSITIONS[k,0], NODE_POSITIONS[k,1],
                   s=size, c=NODE_COLORS[k],
                   alpha=0.8, zorder=3,
                   edgecolors="black", linewidths=0.8)
    buf = BytesIO()
    fig.savefig(buf, format="png", bbox_inches="tight", pad_inches=0, dpi=100)
    plt.close(fig)
    buf.seek(0)
    return np.array(Image.open(buf).convert("RGB").resize((img_size, img_size)))

# 1. Build the SYNTHETIC training set:
# Sequences of (image, counts) with realistic dynamics.
def synth_sequence(rng, length=SEQ_LEN+1, max_count=80):
    """One sequence: SEQ_LEN+1 time steps of 4-region counts."""
    base = rng.uniform(5, 20, size=N_NODES)
    amp = rng.uniform(0, 8, size=N_NODES)
    phase = rng.uniform(0, 2*np.pi, size=N_NODES)
    trend = rng.uniform(-0.5, 0.5, size=N_NODES)
    counts_over_time = []
    for t in range(length):
        c = base + amp*np.sin(2*np.pi*t/12 + phase) + trend*t \
            + rng.normal(0, 1.0, size=N_NODES)
        c = np.clip(np.round(c).astype(int), 0, max_count)

```

```

        counts_over_time.append(c)
    return np.stack(counts_over_time) # (length, 4)

N_SYNTH_SEQ = 800
rng = np.random.default_rng(0)

print(f"Generating {N_SYNTH_SEQ} synthetic sequences...")
seq_images = np.zeros((N_SYNTH_SEQ, SEQ_LEN, IMG_SIZE, IMG_SIZE, 3), dtype=np.uint8)
seq_targets = np.zeros((N_SYNTH_SEQ, N_NODES), dtype=np.float32)

for k in tqdm(range(N_SYNTH_SEQ)):
    counts_seq = synth_sequence(rng, length=SEQ_LEN+1)
    # Past SEQ_LEN images, target = next-step counts
    for t in range(SEQ_LEN):
        seq_images[k, t] = render_count_image(counts_seq[t])
    seq_targets[k] = counts_seq[SEQ_LEN]

# 2. PyTorch setup

# Shape: (N, T, C, H, W)
X = torch.tensor(seq_images).float().permute(0, 1, 4, 2, 3) / 255.0
y = torch.tensor(seq_targets)

perm = torch.randperm(N_SYNTH_SEQ)
split = int(0.85 * N_SYNTH_SEQ)
X_train, y_train = X[perm[:split]], y[perm[:split]]
X_val, y_val = X[perm[split:]], y[perm[split:]]

train_loader = DataLoader(TensorDataset(X_train, y_train), batch_size=16, shuffle=True)
val_loader = DataLoader(TensorDataset(X_val, y_val), batch_size=16)

#3. Model: CNN encoder per frame + LSTM + linear head
class FrameEncoder(nn.Module):
    def __init__(self, embed_dim=64):
        super().__init__()
        self.net = nn.Sequential(
            nn.Conv2d(3, 16, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),
            nn.Conv2d(16, 32, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),
            nn.Conv2d(32, 64, 3, padding=1), nn.ReLU(), nn.AdaptiveAvgPool2d(1),
            nn.Flatten()

```

