In [2]:
```
import os
import numpy as np
import rasterio
from rasterio.windows import Window
from tqdm import tqdm
from PIL import Image

#get the mask and image from the base directores
IMG_TIF = "ChosenContext/Cameroon/raw_data/S2_SouthCameroon_2021_4band.tif"
MASK_TIF = "ChosenContext/Cameroon/raw_data/SouthCameroon_2021_GroundTruth.tif"


OUT_DIR = "Cameroon/preprocessing/processed_dataset"
TILE_SIZE = 512
SCALE_TO_0_1 = True

os.makedirs(f"{OUT_DIR}/images", exist_ok=True)
os.makedirs(f"{OUT_DIR}/masks", exist_ok=True)

#read image and mask
print("Reading image + mask...")
img_ds = rasterio.open(IMG_TIF)
mask_ds = rasterio.open(MASK_TIF)

img = img_ds.read()     # image shape (4, H, W)
mask = mask_ds.read(1)  # mask shape (H, W)

# reorder image to (H, W, 4)
img = np.moveaxis(img, 0, -1)

# scaling 0–1
if SCALE_TO_0_1:
    lo = np.percentile(img[img > 0], 1)
    hi = np.percentile(img, 99)
    img = np.clip((img - lo) / (hi - lo + 1e-8), 0, 1)

H, W, C = img.shape
print("Image shape:", img.shape)
print("Mask shape:", mask.shape)

#split image and mask into identical 512x512 tiles
tile_idx = 0

nrows = int(np.ceil(H / TILE_SIZE))
ncols = int(np.ceil(W / TILE_SIZE))

for r in tqdm(range(nrows)):
    for c in range(ncols):

        y0 = r * TILE_SIZE
        x0 = c * TILE_SIZE

        y1 = min((r+1)*TILE_SIZE, H)
        x1 = min((c+1)*TILE_SIZE, W)

        # create padded tile
        patch_img = np.zeros((TILE_SIZE, TILE_SIZE, 4), dtype=np.float32)
        patch_mask = np.zeros((TILE_SIZE, TILE_SIZE), dtype=np.uint8)

        h = y1 - y0
        w = x1 - x0

        patch_img[:h, :w, :] = img[y0:y1, x0:x1, :]
        patch_mask[:h, :w] = mask[y0:y1, x0:x1]

        # save image and mask
        img_path = f"{OUT_DIR}/images/patch_{r}_{c}.npy"
        mask_path = f"{OUT_DIR}/masks/patch_{r}_{c}.png"

        np.save(img_path, patch_img)

        im = Image.fromarray((patch_mask > 0).astype("uint8") * 255)
        im.save(mask_path)

        tile_idx += 1

print("Done! Created", tile_idx, "tiles.")
```

```
Reading image + mask...
Image shape: (4550, 4190, 4)
Mask shape: (4550, 4190)
100%|██████████████████████████████████████| 9/9 [00:00<00:00, 18.22it/s]
Done! Created 81 tiles.
```

In [3]:
```
import os
import shutil
import numpy as np
from sklearn.model_selection import train_test_split
from PIL import Image

#save path to find images and masks
IMG_DIR = "Cameroon/preprocessing/processed_dataset/images"
MASK_DIR = "Cameroon/preprocessing/processed_dataset/masks"

OUT_BASE = "Cameroon/dataset_split"
TRAIN_IMG = os.path.join(OUT_BASE, "train/images")
TRAIN_MASK = os.path.join(OUT_BASE, "train/masks")
VAL_IMG = os.path.join(OUT_BASE, "val/images")
VAL_MASK = os.path.join(OUT_BASE, "val/masks")
TEST_IMG = os.path.join(OUT_BASE, "test/images")
TEST_MASK = os.path.join(OUT_BASE, "test/masks")

# Make folders
for p in [TRAIN_IMG, TRAIN_MASK, VAL_IMG, VAL_MASK, TEST_IMG, TEST_MASK]:
    os.makedirs(p, exist_ok=True)


#check the image and mask quality
def is_valid_tile(image_path, mask_path, zero_threshold=0.40):
    """
    Returns False if tile should be discarded.
    """

    img = np.load(image_path)

    # Check for NaN, discard immediately
    if np.isnan(img).any():
        return False

    # Count near-zero pixels
    zero_mask = np.all(img < 1e-6, axis=-1)
    zero_ratio = zero_mask.mean()

    if zero_ratio > zero_threshold:
        # too much empty region
        return False

    #  check mask
    mask = np.array(Image.open(mask_path))
    mask_binary = (mask > 127).astype(np.uint8)

    # If mask is basically empty or all-ones ignore
    ratio_ones = mask_binary.mean()
    if ratio_ones < 0.01 or ratio_ones > 0.99:
        return False

    return True


#collect only valid tiles
valid_tiles = []

for f in sorted(os.listdir(IMG_DIR)):
    if not f.endswith(".npy"):
        continue

    img_path = os.path.join(IMG_DIR, f)
    mask_path = os.path.join(MASK_DIR, f.replace(".npy", ".png"))

    if not os.path.exists(mask_path):
        continue

    if is_valid_tile(img_path, mask_path):
        valid_tiles.append(f)

print(f"Total raw tiles: {len(os.listdir(IMG_DIR))}")
print(f"Valid usable tiles: {len(valid_tiles)}")


#split the dataset
train_ratio = 0.60
val_ratio   = 0.24
test_ratio  = 0.05

train_files, other_files = train_test_split(
    valid_tiles, train_size=train_ratio, shuffle=True, random_state=42
)

val_files, test_files = train_test_split(
    other_files,
    train_size=val_ratio / (val_ratio + test_ratio),
    shuffle=True,
    random_state=42
)

print(f"Training:   {len(train_files)}")
print(f"Validation: {len(val_files)}")
print(f"Testing:    {len(test_files)}")

#copy files
def copy_pair(img_list, img_out_dir, mask_out_dir):
    for img_file in img_list:
        shutil.copy(os.path.join(IMG_DIR, img_file), img_out_dir)
        shutil.copy(os.path.join(MASK_DIR, img_file.replace(".npy", ".png")),
                    mask_out_dir)

copy_pair(train_files, TRAIN_IMG, TRAIN_MASK)
copy_pair(val_files, VAL_IMG, VAL_MASK)
copy_pair(test_files, TEST_IMG, TEST_MASK)

print("Dataset split successfully")
```

```
Total raw tiles: 83
Valid usable tiles: 69
Training:   41
Validation: 23
Testing:    5
Dataset split successfully ✔
```

In [ ]: