A Global Satellite Precipitation Model Utilizing Deep Learning with Infrared Data

# **Overview**

Access to high-quality, high-resolution, near-real-time precipitation data is essential for hydrological and meteorological research and disaster mitigation. Though effective, traditional tools such as rain gauges and radar networks have limitations, including sparse coverage in remote areas and high operational costs. Satellite data, with its global coverage and high spatial and temporal resolutions, mitigates these limitations. Satellite precipitation products like Precipitation Estimation from Remotely Sensed Information using Artificial Neural Networks (PERSIANN), Hydro Estimator (HE), Integrated Multi-satellitE Retrievals for Global Precipitation Measurement (IMERG), and Climate Prediction Center MORPHing (CMORPH) utilize both infrared (IR) and passive microwave (PMW) data in their operation. PMW sensors offer detailed atmospheric profiles but suffer from higher latency, whereas IR sensors provide lower latency but only capture cloud-top information. Despite these constraints, IR data remains effective for low-latency precipitation estimation.

Machine learning, especially deep learning, has significantly enhanced precipitation estimation. Convolutional Neural Networks (CNNs), particularly the UNet architecture, have demonstrated high efficiency in this domain. This study introduces a new global deep learning model for precipitation estimation using a UNet model, IR data, and monthly climatological precipitation data, aiming to deliver an operational product with half-hourly temporal resolution and 0.04 x 0.04-degree spatial resolution. Unlike traditional models trained on small areas, which introduce edge noise when combined, the new model employs data processing techniques like image resizing, guided filtering, and quantile mapping to enable training on entire global images. This approach facilitates better learning of precipitation patterns across hemispheres, yielding more consistent results and enhancing applications for near-real-time estimation and climate data record reconstruction.

# **Data**

## ***Input data***

### ***IR Data***

The IR dataset combines 11-micrometer brightness temperature () data from various geostationary satellites, including GMS-5, GOES-8, GOES-10, Meteosat-7, and Meteosat-5. The merging process results in a unified half-hourly infrared field at a spatial resolution of 4 kilometers (Janowiak et al., 2001). This dataset was created by the National Weather Service's Climate Prediction Center. 5000 high-quality images for each month spanning from 2016 to 2021 have been selected and serve as the primary input for generating PUnet precipitation estimates.

### ***Monthly climatological precipitation***

WorldClim 2 precipitation and PERSIANN–Climate Data Record (CDR) are utilized to generate global monthly climatological precipitation and serve as the secondary input for the deep learning model. WorldClim2 is a high-resolution precipitation dataset developed by Fick and Hijmans, derived from spatially interpolated weather station data (Fick and Hijmans, 2017). It offers information on precipitation covering the period from 1970 to 2000 across global land areas. PERSIANN-CDR (Ashouri et al., 2015) is part of the PERSIANN family of precipitation datasets (Nguyen et al., 2019). It provides precipitation data spanning from 1983 to the present with a delay. Implementing monthly climatological precipitation data has been tested and proven to enhance the accuracy of precipitation estimation.

### ***IMERG V07 Final***

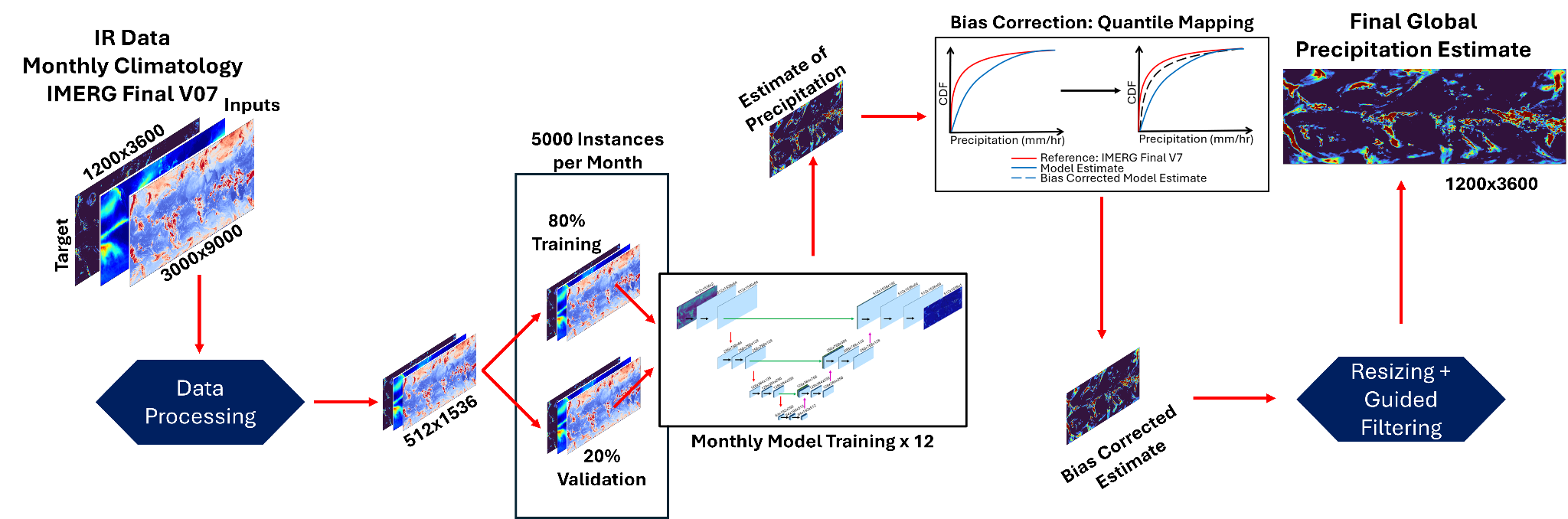
IMERG (Integrated Multi-satellitE Retrievals for Global Precipitation Measurement) Version 07 (V07) represents a significant advancement in global precipitation estimation. Developed by NASA's Global Precipitation Measurement (GPM) mission, IMERG V07 integrates data from multiple satellite sources, including microwave, infrared, and geostationary satellites, to provide high-resolution precipitation estimates on a global scale at half-hourly intervals and a 0.1-degree temporospatial resolution (Huffman et al. 2023). IMERG V07 Final (hereafter referred to as IMERGF), which includes the multi-satellite precipitation estimate with gauge calibration data field for the same period as the infrared data, is used as the target for training the deep learning models and baseline for the global evaluation of daily precipitation estimates from PUnet models.

# **Methodology**

## ***Framework***

The framework in Figure 1 describes the process of integrating satellite imagery, climatological data, and image processing techniques to develop global precipitation estimation models and produce the PUnet estimates. The main steps can be summarized as follows:

1. ***Data Preparation*:** Inputs to the model include infrared (IR) and IMERG Final V07 images captured every 30 minutes between 2016 and 2021, alongside monthly climatological precipitation data sourced from the WorldClim 2 and PERSIANN-CDR datasets. The dataset is processed to select 5,000 high-quality instances (with fewer missing value pixels) for each month, which are then resized to a coarser resolution of 512x1536 pixels. An 80/20 split is used for training and validation purposes.
2. ***Model Training*:** For model training, a global U-Net architecture is employed, with separate training conducted for each of the 12 months under consideration. This approach enables the model to capture seasonal variations in precipitation patterns, ensuring more accurate and reliable estimations throughout the year [Reference needed]. Each monthly U-Net model consists of a total of 7,782,337 parameters and undergoes 100 training epochs to optimize performance.
3. ***Initial Prediction*:** After training the 12 models, both IR images and monthly climatological precipitation data for the desired period are used as input to generate new precipitation estimates. It is crucial that the input data is preprocessed in the same manner as the training data to ensure consistency and accuracy in the predictions.
4. ***Bias Correction*:** The generated precipitation estimates are subjected to additional steps of bias correction to enhance accuracy. The method, Empirical Quantile Mapping (EQM), aligns the probability distribution of the U-Net estimates with the reference IMERGF dataset.
5. ***Resize and Guided Filter*:** To convert the output of the deep learning model (initial size 512x1536) back to the operational resolution of 0.1 degrees (size 1200x3600), the bias-corrected estimates are first resized to 1200x3600 using bilinear interpolation. Following resizing, a guided filter is applied to reduce noise and preserve the cloud-shape information provided by the infrared (IR) data. This step enhances the visual quality of the precipitation estimates, even after the interpolation process.

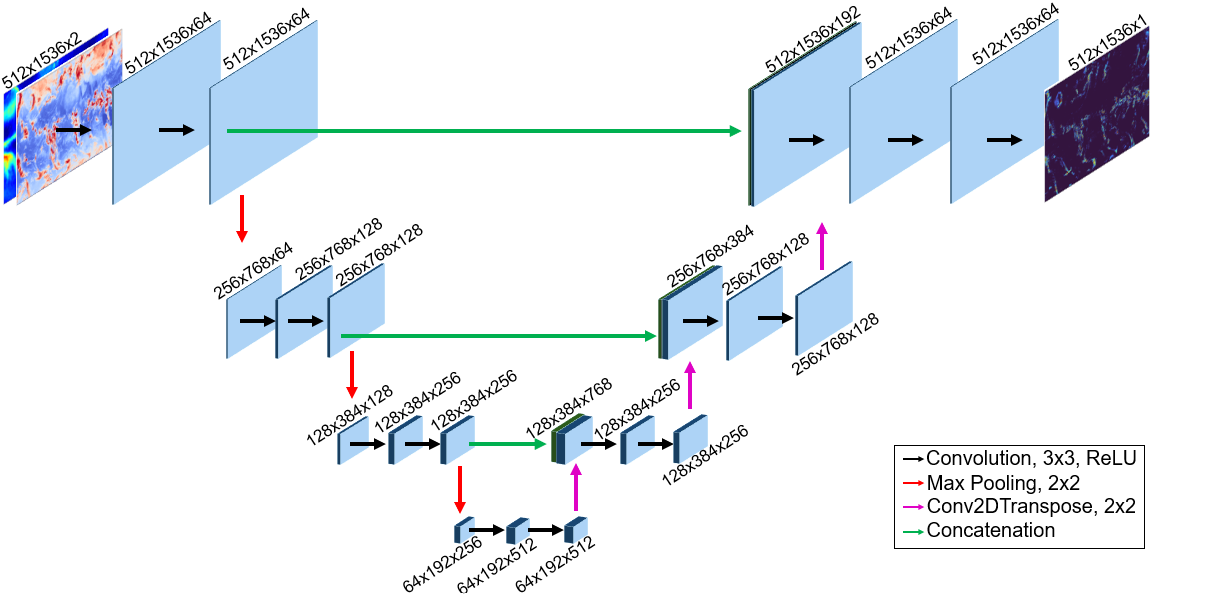


*Figure 1. Flow chart of PERSIANN Global Precipitation Estimation Model using Unet (PUnet)*

Detailed descriptions of the U-Net model, bias correction methods, and guided filtering techniques are provided in the sub-sections below.

## ***UNet Model***

The U-Net architecture features a distinctive U-shaped structure designed as an encoder-decoder framework, enabling the extraction of high-level features while preserving spatial details (Ronneberger et al. 2015). In the encoder section, convolutional and pooling layers progressively reduce the spatial dimensions of the input image, akin to conventional CNNs, to capture context and abstract representations effectively. Conversely, the decoder segment incorporates up-sampling layers to recover spatial information and generate feature maps with pixel-level accuracy (Ji et al. 2018). In the Unet model used for PUnet estimations, the model inputs consist of two 512x1536 matrices containing infrared and monthly climatological precipitation data. These inputs undergo convolution to encode spatial information and extract hierarchical features essential for accurate estimation. Three max-pooling operations are applied during convolution to downsample feature maps while retaining vital information, facilitating the capture of local and global context. Following encoding, convolutional layers are upsampled to match the original input size. Concatenation merges feature maps from the encoding and decoding paths to restore spatial details lost during downsampling. Subsequently, 2D convolutional layers in the decoder refine the upsampled feature maps. Finally, the output layer comprises a single convolutional layer with a ReLU activation function, presenting the predicted image corresponding to the inputs. Mean Squared Error (MSE) serves as the loss function for the Unet model, with a learning rate set at 0.0001. The training process involves 100 steps, with a batch size of 8. The proposed U-Net architecture with details of input shapes, convolutional layers, and operations for global precipitation estimations is depicted in Figure 2.



*Figure 2. UNet architecture for precipitation estimation.*

## ***Bias Correction Technique***

After generating new estimates, bias correction techniques are employed to enhance the accuracy and reliability of the predictions.

**Quantile mapping bias correction**: This study applies empirical quantile mapping (EQM) to align the probability distribution of PUnet estimates with the reference IMERGF data by matching the k'th quantile of the model distribution to the corresponding k'th quantile of the observed data distribution. Data from the training period were used to calculate 20 pre-defined quantiles for both the model and historical distributions. Once the model produces estimates, the mapping process is applied to each pixel to adjust its value according to the calculated quantiles, ensuring consistency with the reference data (Cannon et al. 2015). Mathematically, the EQM correction can be summarized as:

Where refers to bias-corrected precipitation estimate, represents the modeled precipitation value at each grid point, denotes the empirical cumulative distribution function of the modeled data, and is the inverse of the empirical cumulative distribution function of the observed data, also referred to as the quantile function (Cannon et al. 2015).

## ***Guided Filtering***

To further enhance the quality and resolution of the precipitation estimates, guided filtering is employed. This includes resizing the newly estimated precipitation data back to its original resolution of 1200x3600 pixels and performing guided filtering based on high-resolution IR data. This step helps to refine the estimation by smoothing out noise and enhancing spatial coherence, resulting in more accurate and visually appealing precipitation maps.

Guided filtering operates on two images: the input image (1200x3600 image after resizing) and a guidance image (1200x3600 IR data). The goal is to compute an output image that is a smoothed version of while maintaining the structural details provided by .

For each pixel , the output is given by:

Where and are local linear parameters. The parameters and are computed based on the guidance image and the input image within a predefined local window:

Where is the covariance between and , is the variance of , and are the mean values of and within the local window. is a regularization parameter to prevent division by zero and control the smoothness (He et al. 2012). The local window of 5 and of 0.001 are set in the current PUnet operation.

# **Code Package**

Below is the main PERSIANN UNet model package written in Python and Matlab. The package provides all calibrated parameters and functions. There is also a sample data package to run and test the model.

Test\_PUnet.m is the code to plot the model estimates. The binary zip output files contain the 0.1-degree resolution precipitation images with 1200 rows x 3600 columns (60°S to 60°N).

This package has 4 folders and 4 scripts:

1. IR\_data

Contains samples of infrared (IR) data.

1. Parameter

RAIN.npy - Monthly climatological precipitation

Rain1hmax.npy - Monthly max precipitation for bias correction

Obs\_quantiles.npy - Monthly observe quantiles for bias correction quantile mapping

Satellite\_quantiles.npy - Monthly satellite quantile for bias correction quantile mapping

1. Model weight

Includes trained model weights for each of the 12 months

1. PUnet\_output

Stores the precipitation estimate outputs for IR sample data

1. PUnet\_Main\_code.py script

This is the primary code to run the PERSIANN-PUnet model. The code processes input data from the IR\_data and generate global precipitation estimates as .bin.gz files. The current setting generates outputs based on the data available in the IR\_data folder.

1. imresize.py

The image resize function needed for the main code

1. Test\_PUnet.m and loadbfn\_lgz.m

Matlab file to read and visualize the bin.gz files

## ***Main code***

###############################################

# PERSIANN-Unet (PUnet) model for global satellite precipitation estimation using infrared images

# by Phu Nguyen & Vu Dao at Center for Hydrometeorology & Remote Sensing at UC Irvine

# March 2025

###############################################

# Input is merg IR binary .gz file (2, 3298, 9896)

# Import necessary libraries

import cv2

import os

import numpy as np

import gzip

import struct

import glob

# Set the system run on CPUs, and filter out info and warning messages

os.environ['CUDA\_VISIBLE\_DEVICES'] = '-1'

os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '2'

import tensorflow as tf

from tensorflow.keras.layers import Conv2D, BatchNormalization, MaxPooling2D, UpSampling2D, concatenate, Input, Dropout

from tensorflow.keras.models import Model

from scipy.ndimage import uniform\_filter

from cv2.ximgproc import guidedFilter

import concurrent.futures

import tempfile

from tqdm import tqdm

import imresize

###############################################

# Define paths for parameters and input/output data

PARAMETER\_DIR = "./Parameters/"

MODEL\_WEIGHT\_DIR = "./Model weight/"

IR\_DATA\_DIR = "./IR\_data/"

OUTPUT\_DIR = "./PUnet\_output/"

# Define start and end day (format: yyyymmddhh)

startday = 2022010100

endday = 2022010105

# Number of CPU cores for parallel processing

num\_cpus = 6

###############################################

# Functions

# Function to define and load a U-Net model

# It takes input data and a set of weights to produce predictions

def load\_model\_pred(inp, weight):

input\_shape = (512, 1536, 2)

input\_data = Input(shape=input\_shape, name='input\_data')

# Encoder

conv1 = Conv2D(64, (3, 3), padding='same')(input\_data)

conv1 = BatchNormalization()(conv1) #

conv1 = tf.keras.layers.ReLU()(conv1)

conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv1)

pool1 = MaxPooling2D(pool\_size=(2, 2))(conv1)

conv2 = Conv2D(128, (3, 3), padding='same')(pool1)

conv2 = BatchNormalization()(conv2) #

conv2 = tf.keras.layers.ReLU()(conv2)

conv2 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv2)

pool2 = MaxPooling2D(pool\_size=(2, 2))(conv2)

conv3 = Conv2D(256, (3, 3), activation='relu', padding='same')(pool2)

conv3 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv3)

conv3 = Dropout(0.3)(conv3)

pool3 = MaxPooling2D(pool\_size=(2, 2))(conv3)

# Bottleneck

conv4 = Conv2D(512, (3, 3), padding='same')(pool3)

conv4 = BatchNormalization()(conv4)

conv4 = tf.keras.layers.ReLU()(conv4)

conv4 = Dropout(0.4)(conv4)

conv4 = Conv2D(512, (3, 3), activation='relu', padding='same')(conv4)

# Decoder

up3 = UpSampling2D(size=(2, 2))(conv4)

up3 = concatenate([conv3, up3], axis=-1)

conv5 = Conv2D(256, (3, 3), activation='relu', padding='same')(up3)

conv5 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv5)

up2 = UpSampling2D(size=(2, 2))(conv5)

up2 = concatenate([conv2, up2], axis=-1)

conv6 = Conv2D(128, (3, 3), padding='same')(up2)

conv6 = BatchNormalization()(conv6) #

conv6 = tf.keras.layers.ReLU()(conv6)

conv6 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv6)

up1 = UpSampling2D(size=(2, 2))(conv6)

up1 = concatenate([conv1, up1], axis=-1)

conv7 = Conv2D(64, (3, 3), activation='relu', padding='same')(up1)

conv7 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv7)

# Output Layer

output\_data = Conv2D(1, (1, 1), activation='relu')(conv7)

model = Model(inputs=input\_data, outputs=output\_data)

model.set\_weights([weight[f"weight\_{i}"] for i in range(len(weight))])

# Run the model on input

pred = model(inp, training=False)

pred = np.array(pred[0, ..., 0])

tf.keras.backend.clear\_session()

return pred

# Function to clean bad (NaN) pattern in IR images

def removebad(ir0):

ir = np.copy(ir0)

x = np.copy(ir0)

# Identify values less than 0

x = np.where(x < 0, -2.0, 0)

# Create a 2D filter

filter\_size = (100, 200)

x1 = uniform\_filter(x, size=filter\_size, mode='reflect')

ir[x1 < -0.5] = -99

return ir

# Function to fill missing (NaN) in IR images

def fill\_missing(ir\_nan, window\_size=10, mode='reflect'):

nan\_mask = np.isnan(ir\_nan).astype(np.float32)

ir\_filled\_temp = np.where(nan\_mask, 0, ir\_nan)

smoothed = uniform\_filter(ir\_filled\_temp, size=window\_size, mode=mode)

normalization = uniform\_filter(1 - nan\_mask, size=window\_size, mode=mode)

normalization[normalization == 0] = np.nan

filled\_values = smoothed / normalization

ir\_filled = np.where(nan\_mask == 1, filled\_values, ir\_nan)

return ir\_filled

# Function to apply a mapping function for bias correction

def apply\_mapping\_function(future\_satellite\_estimate, satellite\_quantiles, observed\_quantiles):

# Create a mask indicating where interpolation should be performed (i.e., where future\_satellite\_estimate is non-zero)

non\_zero\_mask = future\_satellite\_estimate > 0.001

# Get the indices where the mask is True

lat\_indices, lon\_indices = np.where(non\_zero\_mask)

corrected\_estimate = np.copy(future\_satellite\_estimate)

# Interpolate along the first axis (quantiles) for each non-zero pixel

for lat, lon in zip(lat\_indices, lon\_indices):

# Extract quantile arrays for the current pixel

sat\_quantiles = satellite\_quantiles[:, lat, lon]

obs\_quantiles = observed\_quantiles[:, lat, lon]

value = future\_satellite\_estimate[lat, lon]

# Perform interpolation

if value > sat\_quantiles[-1]:

# Extrapolation for values above the maximum satellite quantile

slope = max((obs\_quantiles[-1] - obs\_quantiles[-2]) / (sat\_quantiles[-1] - sat\_quantiles[-2] + 1e-6), 1.0)

corrected\_estimate[lat, lon] = obs\_quantiles[-1] + slope \* (value - sat\_quantiles[-1])

else:

# Interpolation for values within the satellite quantile range

corrected\_estimate[lat, lon] = np.interp(value, sat\_quantiles, obs\_quantiles)

return corrected\_estimate

# Function to apply multi-scale guided filtering for noise reduction and quality enhancement

def multi\_scale\_guided\_filter(guide, target, base\_radius, base\_epsilon, scales):

filtered = target.copy()

for scale in scales:

scaled\_radius = base\_radius \* scale

scaled\_epsilon = base\_epsilon \* scale\*\*2

filtered = guidedFilter(guide, filtered, int(scaled\_radius), scaled\_epsilon)

return filtered

# Mainfunction to process input data and produce output file

def process\_file(file\_ir):

# Gunzip to read file first

with gzip.open(file\_ir, 'rb') as f\_in:

om = np.frombuffer(f\_in.read(), dtype=np.uint8)

# Reshape the array according to dimensions

# Bin data is scaled by subtracting 75 from the real temperature value, need to +75 to back the real temperature

# 1 file include two timestep at 15 and 45

om = om.astype(np.float32).reshape((2, 3298, 9896))+75

# 255 is the missing value (or 330 after value has been unscaled)

om[om==330]=np.nan

# Processing IR

max\_ir = 300

min\_ir = 173

n\_row = 512

n\_col = 1536

#Loop over 2 timestep of each IR file

for i in range(2):

ir0=om[i,...]

timestep = '15' if i == 0 else '45'

# Clean IR data and resize IR to 1200x3600 for guided filtering and masking NaN

ir=removebad(ir0)

ir[ir <min\_ir] = np.nan

# filling missing 2 time before and after resize to 1200x3600

ir = fill\_missing(ir)

ir = np.concatenate((ir[:, 4948:], ir[:, :4948]), axis=1)

ir = imresize.imresize(ir, output\_shape=[1200,3600],method = 'bilinear')

ir = fill\_missing(ir)

mask = np.where(np.isnan(ir))

# Create guided image for guided filtering

ir1= ir.copy()

ir1[ir1 > max\_ir] = max\_ir

ir1[ir1 < min\_ir] = np.nan

ir1 = (max\_ir - ir1) / (max\_ir - min\_ir) \* 255.0

ir1[ir1 < 0] = np.nan

ir1[np.isnan(ir1)] = 0

# Resize IR to 512x1536, and normalize values for Unet model

ir2 = imresize.imresize(ir, output\_shape=[n\_row,n\_col],method = 'bilinear')

ir2[ir2 > max\_ir] = max\_ir

ir2[ir2 < min\_ir] = np.nan

# Perform the intensity transformation

ir2 = (max\_ir-ir2)/(max\_ir-min\_ir)

idx =np.where(np.isnan(ir2))

ir2[idx]= -1e-5

#Process model input (IR and monthly rain) and run model

ir\_input= ir2[np.newaxis, ...]

inp=np.stack([ir\_input, xtrain2], axis=-1)

inp=inp.astype(np.float32)

pred0 = load\_model\_pred(inp,weight)

# Apply mapping function to prediction

pred = apply\_mapping\_function(pred0,satellite\_quantiles,observed\_quantiles)

pred[pred<0] = 0

# Apply Multi-Scale Guided Filtering

radius = 15 # Adjust the radius of the filtering window as needed

epsilon = 0.0001 # Adjust the regularization parameter as needed

scale\_factors = [1,2,4] # Adjust the radius of the filtering window as needed

rainfall\_emphasis\_factor = 1.1 # Adjust the regularization parameter as needed

rain\_threshold = 0.05 #

guide\_image = ir1.astype(np.float32)

guide\_image = cv2.normalize(guide\_image, None, 0, 1, cv2.NORM\_MINMAX)

# Resize pred after quantile mapping from 512x1536 to 3000x9000

interpolated\_precip\_data = cv2.resize(pred, (3600,1200), interpolation = cv2.INTER\_LINEAR)

interpolated\_precip\_data = interpolated\_precip\_data.astype(np.float32)

norain\_indices = np.where(interpolated\_precip\_data < rain\_threshold)

nan\_indices = np.isnan(interpolated\_precip\_data)

interpolated\_precip\_data[nan\_indices] = 1e-3

weighted\_guide\_image = interpolated\_precip\_data \* guide\_image

weighted\_guide\_image[nan\_indices] = 1e-3

filtered\_result = multi\_scale\_guided\_filter(weighted\_guide\_image, interpolated\_precip\_data, radius, epsilon, scale\_factors)

filtered\_result[mask] = np.nan

# Normalize Filtered Results to Preserve Total Rainfall

total\_rain\_original = np.sum(interpolated\_precip\_data)

filtered\_result = np.maximum(filtered\_result, 1e-6) # Avoid invalid values for power operation

filtered\_result = np.power(filtered\_result, rainfall\_emphasis\_factor)

filtered\_result[norain\_indices] = 0

# Preventing rainfall estimates from exceeding a specified maximum threshold in Rain1hmax

filtered\_result[filtered\_result > Rain1hmax] = Rain1hmax[filtered\_result > Rain1hmax]

# Rescale to Match Total Rainfall

total\_rain\_after\_emphasis = np.sum(filtered\_result)

if total\_rain\_after\_emphasis > 0:

final\_scaling\_factor = total\_rain\_original / total\_rain\_after\_emphasis

else:

final\_scaling\_factor = 1.0

filtered\_result \*= final\_scaling\_factor

# Save to bin.gz file in OUTPUT directory, adjust index based on the input name

file\_path = os.path.join(OUTPUT\_DIR, 'PUnet' + os.path.basename(file\_ir)[5:15]+ timestep +'.bin.gz')

# Convert output to integer format for saving to a binary file.

# Note: When reading the file later, divide by 100 to restore the original value.

pred\_save = np.round(filtered\_result \* 100)

pred\_save[mask]=-9999

pred\_save = pred\_save.astype(np.int16)

with tempfile.NamedTemporaryFile(delete=False, suffix=".bin", mode='wb') as temp\_file:

# Save the array to the temporary binary file

pred\_save.tofile(temp\_file, sep="", format="%<h")

# Compress the temporary binary file into a gzipped binary file

with open(temp\_file.name, 'rb') as temp\_bin\_file, gzip.open(file\_path, 'wb', compresslevel=5) as f:

f.write(temp\_bin\_file.read())

# Remove the temporary binary file

os.remove(temp\_file.name)

return file\_ir[5:15]

###############################################

#Main#

if \_\_name\_\_ == "\_\_main\_\_":

start\_year = int(str(startday)[:4])

end\_year = int(str(endday)[:4])

start\_month = int(str(startday)[4:6])

end\_month = int(str(endday)[4:6])

# Generate unique (year, month) pairs to process

year\_months = []

for year in range(start\_year, end\_year + 1):

for month in range(1, 13):

if (year == start\_year and month < start\_month) or (year == end\_year and month > end\_month):

continue # Skip months outside range

year\_months.append((year, f"{month:02d}")) # Format month as '01', '02', etc.

# Load precomputed quantile and model parameters

observed\_quantiles\_all = np.load(os.path.join(PARAMETER\_DIR, 'Obs\_quantiles.npy'), allow\_pickle=True).item()

satellite\_quantiles\_all = np.load(os.path.join(PARAMETER\_DIR, 'Satellite\_quantiles.npy'), allow\_pickle=True).item()

Rain1hmax\_all = np.load(os.path.join(PARAMETER\_DIR, 'Rain1hmax01.npy'), allow\_pickle=True).item()

RAIN = np.load(os.path.join(PARAMETER\_DIR, 'RAIN.npy'), allow\_pickle=True).item()

for year, month in year\_months:

print(year,month)

# Load month-specific parameters

weight = np.load(os.path.join(MODEL\_WEIGHT\_DIR, f'model\_weight{month}.npy'), allow\_pickle=True).item()

xtrain2 = RAIN.get(month)

observed\_quantiles = observed\_quantiles\_all.get(month)

satellite\_quantiles = satellite\_quantiles\_all.get(month)

Rain1hmax = Rain1hmax\_all.get(month)

# Check IR files within the date range

all\_files = sorted(glob.glob(os.path.join(IR\_DATA\_DIR, f'merg\_{str(year)}{month}\*.gz')))

# Filter files based on startday and endday

filenames = []

for file in all\_files:

basename = os.path.basename(file)

timestamp\_str = basename[5:15] # Extract YYMMDDHH

timestamp = int(f"{timestamp\_str}") # Convert to yyyymmddhh format

if startday <= timestamp <= endday:

filenames.append(file)

print(f"Processing {len(filenames)} files for {year}-{month}")

# Process files only if they exist

if filenames:

with concurrent.futures.ProcessPoolExecutor(max\_workers=num\_cpus) as executor:

total\_files = len(filenames)

futures = {executor.submit(process\_file, fn): fn for fn in filenames}

progress\_bar = tqdm(total=total\_files, desc=f"Processing {year}-{month}", position=0, leave=True)

for future in concurrent.futures.as\_completed(futures):

try:

progress\_bar.update(1)

except Exception as e:

print(f"Exception: {e}")

print(f"Processing complete for files from {startday} to {endday}.")

###############################################