# CAL/VAL Prototype

## README File

**Change log**

June 15, 2025, V02.01.00

* Remade output csv files.
* Improved code efficiency.
* Now all the input can be modified in “sites.csv”.
* Now the region of interest is based on FLEX pixels instead of Sentinel-2 pixels.
* Now the R2 (Coefficient of Determination) is calculated on the linear regression model instead of on true values and predicted values.

May 29, 2025, V02.00.01

* Added all output variables into the table of “Output Variables”

April 08, 2025, V02.00.00

* Added a new python class to calculate several statistical metrics for FLEX and FLOX images.
* Installation & codes instructions improved.

January 07, 2025, V01.01.01

* Improved support for FLEX images.
* Installation & codes instructions improved.

November 29, 2024, V01.01.00

* Add support for FLEX images.

September 2, 2024, V01.00.01

* Add more detailed instructions on how to prepare the files for input
* Add an optional standalone python file (.py) to facilitate the subfolder creation.

August 28, 2024, V01.00.00

* First version of the CAL/VAL Prototype Guideline

**Objective and Scope**

The objective of this CAL/VAL prototype is to calculate coefficients of variation (CV) of vegetation indices inside regions of interest (ROI) centred at field sites, using Sentinel-2 (S2) L2A images and FLEX images, and then compare

First, this prototype reads FLOX input, and it selects all FLEX images which have the same date as FLOX for each site. Then it checks the vegetation pixel percentage inside the ROI to determine whether this FLEX image can be used.

If the vegetation pixel check of a FLEX image is passed, then the prototype will search for a S2 image whose date is closest to the date of the FLEX image but within the input time window days. If there is one S2 image meets the requirements, the prototype will perform a valid pixel check, and only if this check passes, it will calculate the average, standard deviation and coefficient of variation (CV) of vegetation indices (NDVI and NIRvREF) inside the ROI using this S2 image. If the CV is greater than the threshold, then the flag will be set to 0, otherwise the flag will be set to 1.

After the calculation of vegetation indices, the prototype will calculate the average and standard deviation of all SIF bands inside the ROI and then perform a validation with the corresponding FLOX input data. And the prototype will also calculate the SIF correction values using transfer functions and then perform a validation between SIF corrected values and the FLEX SIF values.

In the end the prototype will generate 7 different csv files, each summarizing various aspects of the analysis and validation steps.

**Input variables:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable name** | **Descriptive Name** | **Type** | **Source** | **Remarks** |
| df\_site | .csv file of sites info | Pandas. Dataframe | sites.csv | -Mandatory |
| df\_flox | The FLOX SIF data of all sites | Pandas. Dataframe | flox\_sifparms\_filt\_flextime\_aggr\_avg\_allsites.csv | -Mandatory |
| list\_site\_name | List of names of each site | List | Sites.csv |  |
| list\_site\_lat | List of latitudes of each site | List | Sites.csv |  |
| list\_site\_lon | List of longitudes of each site | List | Sites.csv |  |
| list\_roi | List of reference areas of each site | List | Sites.csv |  |
| list\_threshold\_cv | List of threshold CV of each site | List | Sites.csv |  |
| list\_vegetation\_pixel | List of threshold of vegetation pixel for FLEX images of each site | List | Sites.csv |  |
| list\_threshold\_cloud | List of threshold of valid pixels of each site | List | Sites.csv |  |
| list\_time\_window\_days | List of maximum acceptable time window days of each site | List | Sites.csv |  |

**Output variables:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable name** | **Descriptive Name** | **Type** | **Remarks** |
| list\_flex\_filename | List of filenames of input FLEX images which can be used | List |  |
| list\_flex\_date | List of filenames of input FLEX images which can be used | List |  |
| list\_flex\_time | List of filenames of input FLEX images which can be used | List |  |
| list\_flex\_valid\_pixels | List of percentage of valid pixels of each FLEX image | List |  |
| list\_s2\_filename | List of usable Sentinel-2 image filenames | List |  |
| list\_s2\_date | List of usable Sentinel-2 image dates | List |  |
| list\_s2\_time | List of usable Sentinel-2 image times | List |  |
| list\_s2\_valid\_pixels | List of usable Sentinel-2 image valid pixels percentage | List |  |
| list\_time\_difference | List of usable Sentinel-2 image days difference to corresponding FLEX images | List |  |
| list\_s2\_nirv\_avg | List of average values of NIRvREF of each usable Sentinel-2 image | List |  |
| list\_s2\_nirv\_std | List of standard deviation values of NIRvREF of each usable Sentinel-2 image | List |  |
| list\_s2\_nirv\_cv | List of coefficient of variation values of NIRvREF of each usable Sentinel-2 image | List |  |
| list\_s2\_nirv\_cv\_flag | List of flags of coefficient of variation values of NIRvREF of each usable Sentinel-2 image | List |  |
| list\_s2\_ndvi\_avg | List of average values of NDVI of each usable Sentinel-2 image | List |  |
| list\_s2\_ndvi\_std | List of standard deviation values of NDVI of each usable Sentinel-2 image | List |  |
| list\_s2\_ndvi\_cv | List of coefficient of variation values of NDVI of each usable Sentinel-2 image | List |  |
| list\_s2\_ndvi\_cv\_flag | List of flags of coefficient of variation values of NDVI of each usable Sentinel-2 image | List |  |
| df\_log\_report | Dataframe of log files | Pandas.DataFrame |  |
| df\_sif\_avg | Dataframe of average SIF values of all SIF bands | Pandas.DataFrame |  |
| df\_sif\_std | Dataframe of standard deviation of SIF values of all SIF bands | Pandas.DataFrame |  |
| df\_sif | Dataframe of average SIF values of SIF bands (SIF\_O2A, SIF\_O2B…) | Pandas.DataFrame |  |

**Applicable Version of ATBD**

**Applicable Version of Test Data Set**

* Version: V02.01.00
* Required Inputs:
  + Sites.csv.
  + flox\_sifparms\_filt\_flextime\_aggr\_avg\_allsites.csv
  + All S2 images inside “Input S2 Images” folder should be organized as follows: “Input\_S2\_Images\$SiteName\$filename\”. Change the part $SiteName to the actual names of sites, and $ filename to the actual filenames of your S2 images such as “S2A\_MSIL2A\_20220717T100611\_N0510\_R022\_T32TQQ\_20240717T130024.SAFE”.
  + All FLEX images inside “Input\_FLEX\_Images” folder, organized as follows”Input FLEX Images\Site Name\”.
* Expected Outputs:
  + Full\_Spectrum\_avg\_FLEX\_table.csv
  + Full\_Spectrum\_std\_FLEX\_table.csv
  + L2B\_FLEX\_FLOX\_matchup
  + L2B\_FLEX\_table.csv
  + L2B\_log\_report.csv
  + L2B\_validation\_report\_flex\_flox.csv
  + L2B\_validation\_report\_flex\_tf.csv

**Installation Instructions**

Programming language: Python3.11

Libraries: (Different versions of packages might work as well)

* Numpy >=1.26.4
* pandas >=2.2.2
* shapely >=2.0.5
* geopandas >=1.0.1
* matplotlib >=3.9.1
* lxml >=5.2.2
* beautifulsoup4 >=4.12.3
* GDAL ==3.9.1
* rasterio ==1.3.10
* configparser >=7.1.0
* xarray >=2024.10.0
* scipy >=1.14.1
* netcdf4 >=1.7.2

Steps:

1. Make sure python3 is installed
2. Install the necessary python libraries using pip
   1. pip install -r /path/to/requirements.txt
3. Alternatively, if you are using Anaconda, you can directly create a virtual environment using “environment.yml”. Open cmd.exe via Anaconda and move to the location of the unzipped folder. Then type “conda env create -f environment.yml”.

**Execution Instructions:**

1. Create a virtual python environment using Anaconda or any other similar managers.
2. Ensure you have Python, and the necessary libraries installed in the virtual env. (see Installation Instructions)
3. If you don’t want to add more sites, skip this step and jump to the step 4. Otherwise, open “Sites.csv” and add new lines. The “Site Name” must be unique.
4. For input FLEX images, firstly, you must create subfolders using all available site names inside “Input\_FLEX\_Images”, and then you can put your FLEX images in corresponding subfolders. Make sure the folder structure is identical to “Input Flex Images\IT-SR2\PRS\_TD\_20230616\_101431.nc”.
5. To process S2 images, you also need to put input S2 images inside “Input S2 Images” folder should be organized as follows: “Input\_S2\_Images\$SiteName\$filename\”. Change the part $SiteName to the actual names of sites, and $ filename to the actual filenames of your S2 images such as “S2A\_MSIL2A\_20220717T100611\_N0510\_R022\_T32TQQ\_20240717T130024.SAFE”.
6. Open the “Main.py” in an IDE (such as Spyder or Visual Studio) inside the virtual python environment you have just created, like Spyder or VS Code, and run it.
7. Upon successful execution, the output results can be found inside the folder “Output”.

## Source Code

**Main.py**

# ---------------------------------------------------------------------------- #

#                            Import Python Packages                            #

# ---------------------------------------------------------------------------- #

import os

import time

from datetime import datetime

import shutil

import pandas as pd

# ---------------------------------------------------------------------------- #

#                                 Import Class                                 #

# ---------------------------------------------------------------------------- #

from class\_calval import FLEX, S2

# ---------------------------------------------------------------------------- #

#                                   Main Code                                  #

# ---------------------------------------------------------------------------- #

def main():

    time\_start = time.time()

    print("Code starts!")

    # Initiate classes

    flex = FLEX()

    # --------------------------------- FLOX FILE -------------------------------- #

    if os.path.exists(flex.file\_flox\_csv):

        print("Reading FLOX input!")

        dict\_flox\_dates = flex.check\_flox\_dates()

        print(dict\_flox\_dates)

        time.sleep(1)

        print(f"FLOX input has been read successfully!")

        print("-"\*80)

    else:

        raise FileNotFoundError("The FLOX CSV file is not found! Code aborted!")

    # ------------------------------ LOOP EACH SITE ------------------------------ #

    # Read Sites.csv

    df\_site = flex.get\_site\_info()

    print("Now start to proceed all FLEX images!")

    print("-"\*80)

    # Create empty lists to store the results

    list\_site\_name = []

    list\_site\_lat = []

    list\_site\_lon = []

    list\_roi = []

    list\_threshold\_cv = []

    list\_vegetation\_pixel = []

    list\_threshold\_cloud = []

    list\_time\_window\_days = []

    list\_flex\_date = []

    list\_flex\_time = []

    list\_flex\_filename = []

    list\_flex\_valid\_pixels = []

    list\_s2\_filename = []

    list\_s2\_date = []

    list\_s2\_time = []

    list\_s2\_valid\_pixels = []

    list\_time\_difference = []

    list\_s2\_nirv\_avg = []

    list\_s2\_nirv\_std = []

    list\_s2\_nirv\_cv = []

    list\_s2\_nirv\_cv\_flag = []

    list\_s2\_ndvi\_avg = []

    list\_s2\_ndvi\_std = []

    list\_s2\_ndvi\_cv = []

    list\_s2\_ndvi\_cv\_flag = []

    list\_note = []

    # Some nested lists for empty output

    list\_list\_others = [list\_flex\_date, list\_flex\_time, list\_flex\_filename,

                        list\_flex\_valid\_pixels, list\_s2\_filename, list\_s2\_date,

                        list\_s2\_time, list\_s2\_valid\_pixels, list\_time\_difference,list\_s2\_nirv\_avg,

                        list\_s2\_nirv\_std, list\_s2\_nirv\_cv, list\_s2\_ndvi\_avg, list\_s2\_ndvi\_std,

                        list\_s2\_ndvi\_cv, list\_s2\_nirv\_cv\_flag, list\_s2\_ndvi\_cv\_flag]

    list\_list\_others\_noflex = [list\_s2\_filename, list\_s2\_date,

                    list\_s2\_time, list\_s2\_valid\_pixels, list\_time\_difference,list\_s2\_nirv\_avg,

                    list\_s2\_nirv\_std, list\_s2\_nirv\_cv, list\_s2\_ndvi\_avg, list\_s2\_ndvi\_std,

                    list\_s2\_ndvi\_cv, list\_s2\_nirv\_cv\_flag, list\_s2\_ndvi\_cv\_flag]

    # Iterate each site in Sites.csv!

    for index, row in df\_site.iterrows():

        temp\_start\_time = time.time()

        # Read current line site info

        temp\_site\_name = row['Sites']

        temp\_site\_lat = row['Latitude']

        temp\_site\_lon = row['Longitude']

        temp\_site\_roi = row['ROI']

        temp\_site\_time\_window\_days = row['Time Window Days']

        temp\_site\_threshold\_cv = row['Threshold CV']

        temp\_site\_vegetation\_pixel = row['Vegetation Pixel']

        temp\_site\_threshold\_cloud = row['Threshold Cloud']

        # Save current line site info into the lists

        list\_site\_name.append(temp\_site\_name)

        list\_site\_lat.append(temp\_site\_lat)

        list\_site\_lon.append(temp\_site\_lon)

        list\_roi.append(temp\_site\_roi)

        list\_threshold\_cv.append(temp\_site\_threshold\_cv \* 100)

        list\_vegetation\_pixel.append(temp\_site\_vegetation\_pixel \* 100)

        list\_time\_window\_days.append(temp\_site\_time\_window\_days)

        list\_threshold\_cloud.append(temp\_site\_threshold\_cloud \* 100)

        # Modify the class attributes accordingly

        flex.vegetation\_pixel = temp\_site\_vegetation\_pixel

        # ----------------------------- FLEX IMAGE CHECK ----------------------------- #

        time.sleep(1)

        print("\033[92m" + "\*" \* 5 + "FLEX IMAGES CHECK" + "\*" \* 5 + "\033[0m")

        time.sleep(1)

        # Check if there is a folder inside "input\_flex\_images" for the current site

        temp\_site\_path\_input = os.path.join(flex.path\_flex\_input,temp\_site\_name)

        if not os.path.exists(temp\_site\_path\_input):

            print(f"{temp\_site\_name} doesn't have any input FLEX images! This site has been skipped!")

            print("\033[92m" + "\*" \* 5 + "FLEX IMAGES CHECK DONE" + "\*" \* 5 + "\033[0m")

            print("-"\*80)

            time.sleep(1)

            for temp\_list in list\_list\_others:

                temp\_list.append('N/A')

            list\_note.append('No input FLEX images')

            continue

        # Check if there are any FLEX images inside the folder for the current site

        temp\_site\_flex\_images\_list = os.listdir(temp\_site\_path\_input)

        temp\_site\_flex\_images\_list\_nc = [i for i in temp\_site\_flex\_images\_list if i.endswith('.nc')]

        temp\_site\_flex\_images\_num = len(temp\_site\_flex\_images\_list\_nc)

        if temp\_site\_flex\_images\_num == 0:

            print(f"{temp\_site\_name} doesn't have any input FLEX images! This site has been skipped!")

            print("\033[92m" + "\*" \* 5 + "FLEX IMAGES CHECK DONE" + "\*" \* 5 + "\033[0m")

            print("-"\*80)

            time.sleep(1)

            for temp\_list in list\_list\_others:

                temp\_list.append('N/A')

            list\_note.append('No input FLEX images')

            continue

        # Real work begins here

        print(f"{temp\_site\_name} has {temp\_site\_flex\_images\_num} FLEX image(s)!")

        time.sleep(0.5)

        # Processing all the FLEX images for the current site

        for i in range(temp\_site\_flex\_images\_num):

            temp\_flex\_filename = temp\_site\_flex\_images\_list\_nc[i]

            print(f"Now starting with No.{i + 1} FLEX image '{temp\_flex\_filename}' of the site {temp\_site\_name}")

            # Check FLEX filename format

            flex.check\_filename(temp\_flex\_filename)

            # Get the date of the FLEX image

            temp\_flex\_date = temp\_flex\_filename.split('.')[0].split('\_')[-2]

            if temp\_flex\_date not in dict\_flox\_dates[temp\_site\_name]:

                print(f"The date of the current FLEX image is {temp\_flex\_date}, not found in the FLOX input! This site has been skipped!")

                print("\033[92m" + "\*" \* 5 + "FLEX IMAGES CHECK DONE" + "\*" \* 5 + "\033[0m")

                print("-"\*80)

                time.sleep(1)

                for temp\_list in list\_list\_others:

                    temp\_list.append('N/A')

                list\_note.append(f'No FLOX data on the same date {temp\_flex\_date}')

                continue

            ## Start to process the FLEX image

            print(f"The date of the current FLEX image is {temp\_flex\_date}, found in the FLOX input!!")

            time.sleep(0.5)

            # Veg pixel check - PENDING!!!!!!!!!!

            # if temp\_site\_vegetation\_pixel:

            #     flex.vegetation\_pixel = temp\_site\_vegetation\_pixel

            #     print(f"The vegetation pixel threshold for the site {temp\_site\_name} has been set to {temp\_site\_vegetation\_pixel}!")

            # else:

            #     print("The vegetation pixel threshold is not set, the default value will be used!")

            print("There are enough vegetation pixels inside the ROI in this image! The date and the time of this image will be recorded!")

            time.sleep(0.5)

            # Save the FLEX image information into the lists

            list\_flex\_filename.append(temp\_flex\_filename)

            list\_flex\_date.append(temp\_flex\_filename.split('.')[0].split('\_')[-2])

            list\_flex\_time.append(temp\_flex\_filename.split('.')[0].split('\_')[-1])

            list\_flex\_valid\_pixels.append(100)

            print("\033[92m" + "\*" \* 5 + "FLEX IMAGES CHECK DONE" + "\*" \* 5 + "\033[0m")

            time.sleep(1)

            # ----------------------------- FINDING S2 IMAGE ----------------------------- #

            print("\033[92m" + "\*" \* 5 + "SEARCHING ONE S2 IMAGE WITH THE NEAREST DATE" + "\*" \* 5 + "\033[0m")

            time.sleep(0.5)

            # Now look for S2 images

            print(f"Now looking for the nearest Sentinel-2 image for the site {temp\_site\_name}, within {temp\_site\_time\_window\_days} days!")

            temp\_flex\_image\_datetime = datetime.strptime(temp\_flex\_filename.split('.')[0].split('\_')[-2], '%Y%m%d')

            temp\_path\_s2\_images = os.path.join(flex.path\_s2\_input, temp\_site\_name)

            temp\_s2\_image\_final = os.listdir(temp\_path\_s2\_images)

            if len(temp\_s2\_image\_final) == 0:

                print(f"No Sentinel-2 images found for the site {temp\_site\_name}. This site has been skipped!")

                print("-"\*80)

                time.sleep(1)

                for temp\_list in list\_list\_others\_noflex:

                    temp\_list.append('N/A')

                list\_note.append(f'No input Sentinel-2 images')

                continue

            else:

                print(f"Found {len(temp\_s2\_image\_final)} Sentinel-2 images for the site {temp\_site\_name}!")

                time.sleep(1)

                temp\_s2\_image\_final = temp\_s2\_image\_final[0]

                for j in range(len(os.listdir(temp\_path\_s2\_images))):

                    temp\_s2\_image = os.listdir(temp\_path\_s2\_images)[j]

                    temp\_s2\_image\_date = temp\_s2\_image.split('\_')[2].split('T')[0]

                    temp\_s2\_image\_datetime = datetime.strptime(temp\_s2\_image\_date, '%Y%m%d')

                    if j == 0:

                        temp\_timediff\_final = temp\_s2\_image\_datetime - temp\_flex\_image\_datetime

                    else:

                        temp\_timediff = temp\_s2\_image\_datetime - temp\_flex\_image\_datetime

                        # print(f"Comparing S2 image {temp\_s2\_image} with FLEX image {temp\_flex\_filename}, the time difference is {temp\_timediff}")

                        if abs(temp\_timediff) < abs(temp\_timediff\_final):

                            temp\_timediff\_final = temp\_timediff

                            temp\_s2\_image\_final = temp\_s2\_image

                # Check if the difference between the FLEX image date and the S2 image date is greater than the input time\_window\_days

                if temp\_timediff\_final.days > temp\_site\_time\_window\_days:

                    print(f"The nearest S2 images found for the site {temp\_site\_name} has time difference greater than {temp\_site\_time\_window\_days} days. This site has been skipped!")

                    print("-"\*80)

                    time.sleep(1)

                    for temp\_list in list\_list\_others\_noflex:

                        temp\_list.append('N/A')

                    list\_note.append(f'No input Sentinel-2 images available within {temp\_site\_time\_window\_days} days')

                    continue

                list\_s2\_filename.append(temp\_s2\_image\_final)

                list\_s2\_date.append(temp\_s2\_image\_final.split('\_')[2].split('T')[0])

                list\_s2\_time.append(temp\_s2\_image\_final.split('\_')[2].split('T')[1])

                list\_time\_difference.append(temp\_timediff\_final.days)

                print(f"S2 image '{temp\_s2\_image\_final}' has the nearest date ({temp\_timediff\_final.days} days) to the FLEX image {temp\_flex\_filename}")

                time.sleep(0.5)

                print(f"Now reading the metadata of the S2 image......")

                # Finally we can initiate the S2 class provided we already find the S2 image to use!

                s2 = S2(temp\_site\_name, temp\_site\_lat, temp\_site\_lon, temp\_s2\_image\_final)

                s2.area = temp\_site\_roi

                s2.threshold\_cv = temp\_site\_threshold\_cv

                s2.cloud = temp\_site\_threshold\_cloud

                s2.flex\_filename = temp\_flex\_filename

                print("\033[92m" + "\*" \* 5 + "SEARCHING ONE S2 IMAGE WITH THE NEAREST DATE DONE" + "\*" \* 5 + "\033[0m")

                time.sleep(1)

                # ------------------------------ S2 IMAGE CHECK ------------------------------ #

                print("\033[92m" + "\*" \* 5 + "S2 Valid Pixel Check" + "\*" \* 5 + "\033[0m")

                time.sleep(1)

                # Create the cache subfolder for the current site

                s2.create\_cache\_subfolder(temp\_site\_name)

                print(f"Checking the valid pixels of the S2 image. Only if the valid pixels are greater than {temp\_site\_threshold\_cloud \* 100}% of the total pixels, the S2 image will be used for further processing!")

                # Read masks of opaque clouds, cirrus clouds and snow ice areas

                temp\_pass\_l2a, temp\_valid\_pixels\_l2a, temp\_valid\_pixels\_percentage\_l2a = s2.cal\_valid\_pixels()

                # Save valid pixel result

                list\_s2\_valid\_pixels.append(temp\_valid\_pixels\_percentage\_l2a \* 100)

                time.sleep(1)

                # Valid pixels check

                if temp\_pass\_l2a:

                    print(f"{temp\_site\_name} and its S2 image {temp\_s2\_image\_final} has sufficient valid pixels!")

                    print("\033[92m" + "\*" \* 5 + "S2 Valid Pixel Check DONE" + "\*" \* 5 + "\033[0m")

                    time.sleep(1)

                    # ------------------------------ S2 NDVI NIRvREF ----------------------------- #

                    print("\033[92m" + "\*" \* 5 + "S2 NDVI & NIRvREF Calculation" + "\*" \* 5 + "\033[0m")

                    print(f"Now calculating NDVI and NIRvREF inside the ROI of the site {temp\_site\_name}......")

                    temp\_ndvi\_std, temp\_ndvi\_avg, temp\_ndvi\_cv, temp\_ndvi\_flag, temp\_nirv\_std, temp\_nirv\_avg, temp\_nirv\_cv, temp\_nirv\_flag = s2.cal\_l2a\_indices()

                    list\_s2\_ndvi\_std.append(temp\_ndvi\_std)

                    list\_s2\_ndvi\_avg.append(temp\_ndvi\_avg)

                    list\_s2\_ndvi\_cv.append(temp\_ndvi\_cv)

                    list\_s2\_ndvi\_cv\_flag.append(temp\_ndvi\_flag)

                    list\_s2\_nirv\_std.append(temp\_nirv\_std)

                    list\_s2\_nirv\_avg.append(temp\_nirv\_avg)

                    list\_s2\_nirv\_cv.append(temp\_nirv\_cv)

                    list\_s2\_nirv\_cv\_flag.append(temp\_nirv\_flag)

                    print("\033[92m" + "\*" \* 5 + "S2 NDVI & NIRvREF Calculation DONE" + "\*" \* 5 + "\033[0m")

                    # --------------------------------- FLEX SIF --------------------------------- #

                    print("\033[92m" + "\*" \* 5 + "FLEX SIF Calculation" + "\*" \* 5 + "\033[0m")

                    time.sleep(0.5)

                    print(f"Now starting to calculate the SIF for the site {temp\_site\_name} and its FLEX image {temp\_flex\_filename}!")

                    # Read converted FLEX .tiff file from the cache folder

                    flex.cal\_sif(temp\_site\_name, temp\_flex\_filename, temp\_site\_lon, temp\_site\_lat, temp\_site\_roi, temp\_s2\_image\_final)

                    flex.sif\_output(temp\_site\_name, temp\_flex\_filename, temp\_site\_lon, temp\_site\_lat, temp\_site\_roi, temp\_s2\_image\_final)

                    print("\033[92m" + "\*" \* 5 + "FLEX SIF Calculation DONE" + "\*" \* 5 + "\033[0m")

                    time.sleep(1)

                    # ----------------------------- Transfer Function ---------------------------- #

                    print("\033[92m" + "\*" \* 5 + "TRANSFER FUNCTION" + "\*" \* 5 + "\033[0m")

                    time.sleep(0.5)

                    print(f"Now applying transfer functions for the site {temp\_site\_name} and its FLEX image {temp\_flex\_filename}!")

                    bool\_flox\_invalid = s2.cal\_transfer\_function(temp\_flex\_date)

                    if bool\_flox\_invalid:

                        list\_note.append('FLOX is on an invalid pixel')

                    else:

                        list\_note.append("N/A")

                    print("\033[92m" + "\*" \* 5 + "TRANSFER FUNCTION DONE" + "\*" \* 5 + "\033[0m")

                    temp\_end\_time = time.time()

                    temp\_elapsed\_time = temp\_end\_time - temp\_start\_time

                    print(f"The calculation and validation of site {temp\_site\_name} has been finished successfully, which took {temp\_elapsed\_time:.2f} seconds! ")

                    time.sleep(0.5)

                    print("-"\*80)

                    s2.remove\_cache()

                    time.sleep(1)

                else:

                    print(f"\033[92mThe calculation and validation of site {temp\_site\_name} and its S2 image {temp\_s2\_image\_final} has been skipped, due to exceeding invalid pixels!\033[0m")

                    list\_s2\_nirv\_avg.append('N/A')

                    list\_s2\_nirv\_std.append('N/A')

                    list\_s2\_nirv\_cv.append('N/A')

                    list\_s2\_nirv\_cv\_flag.append('N/A')

                    list\_s2\_ndvi\_avg.append('N/A')

                    list\_s2\_ndvi\_std.append('N/A')

                    list\_s2\_ndvi\_cv.append('N/A')

                    list\_s2\_ndvi\_cv\_flag.append('N/A')

                    list\_note.append(f"The percentage of invalid pixels exceeding {s2.cloud \* 100}%")

                    print("-"\*80)

                    time.sleep(1)

                    continue

    # Loop finished, now we save the output to a new .csv file

    # ---------------------------------- output ---------------------------------- #

    # log report

    df\_log\_report = pd.DataFrame({

        "site\_code": list\_site\_name,

        "latitude": list\_site\_lat,

        "longitude": list\_site\_lon,

        "reference\_area": list\_roi,

        "time\_window": list\_time\_window\_days,

        "threshold\_CV": list\_threshold\_cv,

        "vegetation\_pixel": list\_vegetation\_pixel,

        "threshold\_cloud": list\_threshold\_cloud,

        "flex\_date": list\_flex\_date,

        "flex\_time": list\_flex\_time,

        "flex\_filename": list\_flex\_filename,

        "flex\_valid\_pixels": list\_flex\_valid\_pixels,

        "s2\_filename": list\_s2\_filename,

        "s2\_date": list\_s2\_date,

        "s2\_time": list\_s2\_time,

        "time\_difference\_s2\_flex": list\_time\_difference,

        "s2\_valid\_pixels": list\_s2\_valid\_pixels,

        "s2\_ndvi\_avg": list\_s2\_ndvi\_avg,

        "s2\_ndvi\_sd": list\_s2\_ndvi\_std,

        "s2\_ndvi\_cv": list\_s2\_ndvi\_cv,

        "s2\_ndvi\_cv\_flag": list\_s2\_ndvi\_cv\_flag,

        "s2\_nirv\_avg": list\_s2\_nirv\_avg,

        "s2\_nirv\_sd": list\_s2\_nirv\_std,

        "s2\_nirv\_cv": list\_s2\_nirv\_cv,

        "s2\_nirv\_cv\_flag": list\_s2\_nirv\_cv\_flag,

        "note": list\_note

    })

    df\_log\_report.to\_csv(os.path.join(s2.path\_output,"L2B\_log\_report.csv"), index = False)

    # sif avg

    list\_csv\_file\_avg = []

    for csv\_file in os.listdir(os.path.join(s2.path\_cache, 'FLEX', 'avg')):

        if csv\_file.endswith('.csv'):

            list\_csv\_file\_avg.append(os.path.join(s2.path\_cache,'FLEX', 'avg', csv\_file))

    df\_sif\_avg = pd.concat([pd.read\_csv(f) for f in list\_csv\_file\_avg], ignore\_index=True)

    df\_sif\_avg.to\_csv(os.path.join(s2.path\_output, "Full\_Spectrum\_avg\_FLEX\_table.csv"), index=False)

    # sif std

    list\_csv\_file\_std = []

    for csv\_file in os.listdir(os.path.join(s2.path\_cache, 'FLEX', 'std')):

        if csv\_file.endswith('.csv'):

            list\_csv\_file\_std.append(os.path.join(s2.path\_cache,'FLEX', 'std', csv\_file))

    df\_sif\_std = pd.concat([pd.read\_csv(f) for f in list\_csv\_file\_std], ignore\_index=True)

    df\_sif\_std.to\_csv(os.path.join(s2.path\_output, "Full\_Spectrum\_std\_FLEX\_table.csv"), index=False)

    # sif

    list\_csv\_file = []

    for csv\_file in os.listdir(os.path.join(s2.path\_cache, 'FLEX', 'sif')):

        if csv\_file.endswith('.csv'):

            list\_csv\_file.append(os.path.join(s2.path\_cache,'FLEX', 'sif', csv\_file))

    df\_sif = pd.concat([pd.read\_csv(f) for f in list\_csv\_file], ignore\_index=True)

    df\_sif.to\_csv(os.path.join(s2.path\_output, "L2B\_FLEX\_table.csv"), index=False)

    flex.create\_matchup\_report()

    flex.cal\_statistic\_flex\_flox()

    flex.cal\_statistic\_flex\_tf()

    # Delete cache folder?

    if s2.bool\_delete\_cache:

        shutil.rmtree(s2.path\_cache)

        print("The cache folder and all its contents has been deleted permanently! ")

    print(f"Please find the final output.csv in the following folder: {s2.path\_output}")

    # ------------------------------ Code Terminates ----------------------------- #

    time\_end = time.time()

    time\_elapsed = time\_end - time\_start

    print(f"This python code has finished its work, and in totale it has taken {time\_elapsed:.2f} seconds!")

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

    main()

**class\_calval.py**

import os

import csv

from typing import Optional, Union

import re

import numpy as np

import pandas as pd

import xarray as xr

from bs4 import BeautifulSoup

import shapely as shp

import geopandas as gpd

import rasterio as rio

import rasterio.mask

from sklearn.metrics import r2\_score

from scipy.stats import linregress

from sklearn.linear\_model import LinearRegression

class CalVal:

    # Constuctor

    def \_\_init\_\_(self):

        '''

        Initialize the class.

        '''

        # -------------------------------- Attributes -------------------------------- #

        # Current work directory (where the script is located)

        self.\_path\_main = os.path.realpath(os.path.dirname(\_\_file\_\_))

        # Input folder

        self.\_path\_input = os.path.join(self.path\_main, "input")

        # The absolute path of the S2 images

        self.\_path\_s2\_input = os.path.join(self.\_path\_main,"input\_s2\_images")

        # The absolute path of the FLEX images

        self.\_path\_flex\_input = os.path.join(self.path\_main,"input\_flex\_images")

        # Output folder

        self.\_path\_output = os.path.join(self.path\_main, "output")

        # FLOX input

        self.\_file\_flox\_csv = os.path.join(self.path\_input,'flox\_sifparms\_filt\_flextime\_aggr\_avg\_allsites.csv')

        # The absolute path to the input .csv file, where the info of all sites are saved.

        self.\_file\_site\_csv = os.path.join(self.path\_input, "sites.csv")

        # The absolute path to the folder, where interim files are saved.

        self.\_path\_cache = os.path.join(self.path\_main, "cache")

        # A boolean variable which determines whether the cache files will be deleted unpon the completion of the code. False by default.

        self.\_bool\_delete\_cache = False

        # Flex filename

        self.flex\_filename = None

        ### Automatically check

        self.\_\_check\_site\_csv()

        self.\_\_check\_output()

        self.\_\_check\_cache()

    # ------------------------------ Private Methods ----------------------------- #

    # Create an output folder if not exists

    def \_\_check\_output(self):

        if not os.path.exists(self.path\_output):

            print("No output folder found! Creating a new output folder......")

            os.makedirs(self.path\_output)

            print(f"Output folder {self.path\_output} created successfully!")

    # Check if "Sites.csv" exists. Otherwise gives error.

    def \_\_check\_site\_csv(self):

        if not os.path.exists(self.file\_site\_csv):

            raise FileNotFoundError(f"The input directory {self.path\_input} doesn't contain the 'Sites.csv' file!")

        # else:

        #     print(f"'Sites.csv' file has been found in {self.\_path\_Main}!")

    # Create a cache folder if not exists

    def \_\_check\_cache(self):

        if not os.path.exists(self.\_path\_cache):

            os.makedirs(self.\_path\_cache)

    # ------------------------------ Getter & Setter ----------------------------- #

    @property

    def path\_main(self):

        return self.\_path\_main

    @property

    def path\_input(self):

        return self.\_path\_input

    @property

    def path\_s2\_input(self):

        return self.\_path\_s2\_input

    @property

    def path\_flex\_input(self):

        return self.\_path\_flex\_input

    @property

    def path\_output(self):

        return self.\_path\_output

    @property

    def file\_site\_csv(self):

        return self.\_file\_site\_csv

    @property

    def path\_cache(self):

        return self.\_path\_cache

    @property

    def bool\_delete\_cache(self):

        return self.\_bool\_delete\_cache

    @property

    def file\_flox\_csv(self):

        return self.\_file\_flox\_csv

    @file\_flox\_csv.setter

    def file\_flox\_csv(self, value):

        self.\_file\_flox\_csv = value

    # ------------------------------ Public Methods ------------------------------ #

    # Create a pandas dataframe using Sites.csv

    def get\_site\_info(self):

        df\_sites = pd.read\_csv(self.file\_site\_csv)

        # Site names

        site\_name = df\_sites["site\_code"]

        if not site\_name.notna().all():

            raise ValueError("Please make sure there is no missing site name in the .csv file!")

        # Convert site names to string

        site\_name\_str = [str(element) for element in site\_name]

        # Site lat

        site\_lat = df\_sites["latitude"]

        if not site\_lat.notna().all():

            raise ValueError("Please make sure there is no missing latitude in the .csv file!")

        if not pd.to\_numeric(site\_lat, errors='coerce').notna().all():

            raise ValueError("Please make sure latitudes are numeric values in the .csv file!")

        if not site\_lat.between(-90, 90).all():

            raise ValueError("Please make sure latitudes are within the range of -90 to 90!")

        # Site lon

        site\_lon = df\_sites["longitude"]

        if not site\_lon.notna().all():

            raise ValueError("Please make sure there is no missing latitude in the .csv file!")

        if not pd.to\_numeric(site\_lon, errors='coerce').notna().all():

            raise ValueError("Please make sure longtitudes are numeric values in the .csv file!")

        if not site\_lon.between(-180, 180).all():

            raise ValueError("Please make sure longtitudes are within the range of -180 to 180!")

        # reference\_area

        site\_roi = df\_sites["reference\_area(m)"]

        # time\_window\_days

        site\_time\_window = df\_sites["time\_window(days)"]

        # threshold\_cv

        site\_threshold\_cv = df\_sites["threshold\_CV(%)"] / 100

        # vegetation\_pixel

        site\_vegetation\_pixel = df\_sites["vegetation\_pixel(%)"] / 100

        # threshold\_cloud

        site\_threshold\_cloud = df\_sites["threshold\_cloud(%)"] / 100

        # Create final CSV

        df\_sites = pd.DataFrame({

            "Sites": site\_name\_str,

            "Latitude": site\_lat,

            "Longitude": site\_lon,

            "ROI": site\_roi,

            "Time Window Days": site\_time\_window,

            "Threshold CV": site\_threshold\_cv,

            "Vegetation Pixel": site\_vegetation\_pixel,

            "Threshold Cloud": site\_threshold\_cloud

        })

        print("'Sites.csv' read successfully!")

        return df\_sites

    def create\_matchup\_report(self):

        # Merge transfer function output

        df\_tf = pd.concat((pd.read\_csv(os.path.join(self.path\_cache,'TF',csv\_file)) for csv\_file in os.listdir(os.path.join(self.path\_cache,'TF'))), ignore\_index=True)

        df\_tf['date'] = df\_tf['date'].astype(str)

        df\_tf.rename(columns={

            "SIF\_O2A": "SIF\_O2A\_tf",

            "SIF\_FARRED\_max": "SIF\_FARRED\_max\_tf",

            "SIF\_int": "SIF\_int\_tf",

            "SIF\_O2B": "SIF\_O2B\_tf",

            "SIF\_RED\_max": "SIF\_RED\_max\_tf",

            "SIF\_O2A\_un": "SIF\_O2A\_un\_tf",

            "SIF\_FARRED\_max\_un": "SIF\_FARRED\_max\_un\_tf",

            "SIF\_int\_un": "SIF\_int\_un\_tf",

            "SIF\_O2B\_un": "SIF\_O2B\_un\_tf",

            "SIF\_RED\_max\_un": "SIF\_RED\_max\_un\_tf"

        }, inplace = True)

        # Read FLOX

        df\_flox = pd.read\_csv(self.file\_flox\_csv, sep = ';')

        df\_flox = df\_flox[['ID\_SITE','UTC\_datetime','SIF\_FARRED\_max','SIF\_FARRED\_max\_wvl','SIF\_RED\_max','SIF\_RED\_max\_wvl','SIF\_O2B','SIF\_O2A','SIF\_int','SIF\_FARRED\_max\_un','SIF\_FARRED\_max\_wvl\_un','SIF\_RED\_max\_un','SIF\_RED\_max\_wvl\_un','SIF\_O2B\_un','SIF\_O2A\_un','SIF\_int\_un']]

        df\_flox.rename(columns={'ID\_SITE': 'site\_code', 'UTC\_datetime': 'date'}, inplace=True)

        df\_flox['date'] = pd.to\_datetime(df\_flox['date'], format='%d/%m/%Y %H:%M')

        # Convert to string date format

        df\_flox['date'] = df\_flox['date'].dt.strftime('%Y%m%d')

        # Read FLEX

        df\_flex = pd.read\_csv(os.path.join(self.path\_output,"L2B\_FLEX\_table.csv"))

        df\_flex = df\_flex[['site\_code','flex\_date','SIF\_FARRED\_max','SIF\_FARRED\_max\_wvl','SIF\_RED\_max','SIF\_RED\_max\_wvl','SIF\_O2B','SIF\_O2A','SIF\_int','SIF\_FARRED\_max\_un','SIF\_FARRED\_max\_wvl\_un','SIF\_RED\_max\_un','SIF\_RED\_max\_wvl\_un','SIF\_O2B\_un','SIF\_O2A\_un','SIF\_int\_un']]

        df\_flex.rename(columns={'flex\_date': 'date'}, inplace=True)

        df\_flex['date'] = df\_flex['date'].astype(str)

        # Merge into a single dataframe

        df\_merge = pd.merge(df\_flox,df\_flex,how='inner',on=['site\_code','date'], suffixes=('\_flox','\_flex'))

        df\_merge = pd.merge(df\_merge,df\_tf,how='inner',on=['site\_code','date'])

        df\_merge.to\_csv(os.path.join(self.path\_output,"L2B\_FLEX\_FLOX\_matchup.csv"), index=False)

class FLEX(CalVal):

    # FLEX image resolution

    \_FLEX\_RESOLUTION = 300

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        # Input FLEX Images path

        # ROI size

        self.\_area\_roi = 900

        # Vegetation pixel percentage!

        self.\_vegetation\_pixel = 0.5

        # Check input flex images folder

        self.\_\_check\_input()

    # ------------------------------ Private Methods ----------------------------- #

    def \_\_check\_input(self):

        if not os.path.exists(self.path\_flex\_input):

            raise FileNotFoundError(f"The working directory {self.\_path\_main} doesn't contain the 'Input FLEX Images' Folder!")

        if not bool(os.listdir(self.path\_flex\_input)):

            raise FileNotFoundError(f"There is no FLEX image found inside the 'Input FLEX Images' folder!")

    # ----------------------------- Getter and Setter ---------------------------- #

    # Getter and setter for ROI

    @property

    def area\_roi(self):

        return self.\_area\_roi

    @area\_roi.setter

    def area\_roi(self, value):

        if value < self.FLEX\_RESOLUTION:

            raise ValueError("The ROI must be greater than 300m x 300m!")

        if value % self.FLEX\_RESOLUTION != 0:

            raise ValueError("The ROI must contain complete FLEX pixel(s)!")

        self.\_area\_ROI = value

    # Getter for FLEX image resolution; no setter since it is a constant!

    @property

    def FLEX\_RESOLUTION(self):

        return self.\_FLEX\_RESOLUTION

    # Getter for vegetation pixel

    @property

    def vegetation\_pixel(self):

        return self.\_vegetation\_pixel

    @vegetation\_pixel.setter

    def vegetation\_pixel(self, value):

        if value < 0 or value > 1:

            raise ValueError("The valid vegetation pixel percentage must be between 0 and 1!")

        self.\_vegetation\_pixel = value

    # ------------------------------ Public Methods ------------------------------ #

    ## Check file name convention

    # PRS\_TD\_20230616\_101431.nc

    def check\_filename(self, filename: str) -> None:

        '''

        A function used to check whether the filename of a FLEX image corresponds to the correct format. If not, it will raise an error.

        '''

        if not re.fullmatch(r"^PRS\_TD\_\d{8}\_\d{6}\.nc$", filename):

            raise ValueError(f"The filename '{filename}' is not correct! It should have the same format as 'PRS\_TD\_20230616\_101431.nc'!")

    ## Check FLOX dates

    def check\_flox\_dates(self) -> dict:

        # Read FLOX CSV input file

        df\_flox = pd.read\_csv(self.file\_flox\_csv, sep=';')

        # Regulate datetime format

        df\_flox['UTC\_datetime'] = pd.to\_datetime(df\_flox['UTC\_datetime'], format='%d/%m/%Y %H:%M')

        # Convert to string date format

        df\_flox['UTC\_datetime'] = df\_flox['UTC\_datetime'].dt.strftime('%Y%m%d')

        # Remove useless columns

        df\_flox = df\_flox[['ID\_SITE','UTC\_datetime']]

        # Export

        df\_flox\_dict = df\_flox.groupby('ID\_SITE')['UTC\_datetime'].apply(list).to\_dict()

        return df\_flox\_dict

    ## SIF Calculation

    def cal\_sif(self, site\_name: str, filename: str, site\_lon: Union[int, float], site\_lat: Union[int, float], roi: int, s2\_filename: str) -> None:

        # Open the FLEX image

        temp\_ds = xr.open\_dataset(os.path.join(self.path\_flex\_input,site\_name,filename))

        # Read longitudes and latitudes from the dataset

        longitudes = temp\_ds['longitude'].values

        latitudes = temp\_ds['latitude'].values

        # Get the indices of the closest longitudes and latitudes to the site

        lon\_left = np.where(longitudes <= site\_lon)[0][-1]

        lon\_right = np.where(longitudes >= site\_lon)[0][0]

        lat\_top = np.where(latitudes >= site\_lat)[0][-1]

        lat\_bottom = np.where(latitudes <= site\_lat)[0][0]

        # print(f"lon\_left: {lon\_left}, lon\_right: {lon\_right}, lat\_top: {lat\_top}, lat\_bottom: {lat\_bottom}")

        # Now find the indices of the pixel where the site is located

        if abs(site\_lon - longitudes[lon\_left]) < abs(site\_lon - longitudes[lon\_right]):

            lon\_index = lon\_left

        else:

            lon\_index = lon\_right

        if abs(site\_lat - latitudes[lat\_top]) < abs(site\_lat - latitudes[lat\_bottom]):

            lat\_index = lat\_top

        else:

            lat\_index = lat\_bottom

        # Prepare empty lists to store results of each loop

        temp\_list\_sif\_name = []

        temp\_list\_sif\_avg = []

        temp\_list\_sif\_std = []

        temp\_name\_rar = list(temp\_ds.data\_vars)

        for var\_name in temp\_name\_rar:

            if "Sif Emission Spectrum\_sif\_wavelength\_grid" in var\_name:

                if roi == 300:

                    temp\_array = temp\_ds[var\_name][lat\_index,lon\_index].values

                elif roi == 600:

                    if abs(site\_lat - latitudes[lat\_index - 1]) < abs(site\_lat - latitudes[lat\_index + 1]):

                        lat\_index\_600 = lat\_index - 1

                    else:

                        lat\_index\_600 = lat\_index + 1

                    if abs(site\_lon - longitudes[lon\_index - 1]) < abs(site\_lon - longitudes[lon\_index + 1]):

                        lon\_index\_600 = lon\_index - 1

                    else:

                        lon\_index\_600 = lon\_index + 1

                    lat\_index\_start = min(lat\_index\_600, lat\_index)

                    lat\_index\_end = max(lat\_index\_600, lat\_index)

                    lon\_index\_start = min(lon\_index\_600, lon\_index)

                    lon\_index\_end = min(lon\_index\_600, lon\_index)

                    temp\_array = temp\_ds[var\_name][lat\_index\_start:(lat\_index\_end+1),lon\_index\_start:(lon\_index\_end+1)].values

                elif roi == 900:

                    temp\_array = temp\_ds[var\_name][(lat\_index-1):(lat\_index+2),(lon\_index-1):(lon\_index+2)].values

                else:

                    temp\_array = temp\_ds[var\_name][(lat\_index-1):(lat\_index+2),(lon\_index-1):(lon\_index+2)].values

                    print(f"The ROI {roi} m x {roi} m is greater than 900m x 900m. The code only supports 300m, 600m and 900m ROI. So now the calculation will be performed on a 900m ROI. ")

                temp\_list\_sif\_name.append(var\_name)

                temp\_avg = np.average(temp\_array).item()

                temp\_list\_sif\_avg.append(temp\_avg)

                temp\_std = np.std(temp\_array).item()

                temp\_list\_sif\_std.append(temp\_std)

        temp\_list\_sif\_name = ['site\_code','latitude','longitude','flex\_date','flex\_time','flex\_filename','s2\_filename'] + temp\_list\_sif\_name

        temp\_list\_sif\_avg = [site\_name, site\_lat, site\_lon, filename.split('.')[0].split('\_')[-2], filename.split('.')[0].split('\_')[-1], filename, s2\_filename] + temp\_list\_sif\_avg

        temp\_list\_sif\_std = [site\_name, site\_lat, site\_lon, filename.split('.')[0].split('\_')[-2], filename.split('.')[0].split('\_')[-1], filename, s2\_filename] + temp\_list\_sif\_std

        temp\_df\_sif\_avg = pd.DataFrame([temp\_list\_sif\_avg], columns = temp\_list\_sif\_name)

        temp\_df\_sif\_std = pd.DataFrame([temp\_list\_sif\_std], columns = temp\_list\_sif\_name)

        if not os.path.exists(os.path.join(self.path\_cache,'FLEX', 'avg')):

            os.makedirs(os.path.join(self.path\_cache,'FLEX', 'avg'))

        if not os.path.exists(os.path.join(self.path\_cache,'FLEX', 'std')):

            os.makedirs(os.path.join(self.path\_cache,'FLEX', 'std'))

        temp\_df\_sif\_avg.to\_csv(os.path.join(self.path\_cache,'FLEX','avg',site\_name + "\_" + filename + ".csv"), index = False)

        temp\_df\_sif\_std.to\_csv(os.path.join(self.path\_cache,'FLEX','std',site\_name + "\_" + filename + ".csv"), index = False)

    def sif\_output(self, site\_name: str, filename: str, site\_lon: Union[int, float], site\_lat: Union[int, float], roi: int, s2\_filename: str) -> list:

        '''

        This function is used to calculate average values of a series of SIF metrics in a 3x3 pixel ROI of a FLEX image of a site.

        Parameters:

        - site\_name: str, the name of the site

        - filename: str, the name of the FLEX image

        - site\_lon: Union[int, float], the longitude of the site

        - site\_lat: Union[int, float], the latitude of the site

        Returns:

            list\_value: list, a list of average values of SIF metrics in a 3x3 pixel ROI of a FLEX image of a site:

        '''

        # Open the FLEX image

        temp\_ds = xr.open\_dataset(os.path.join(self.path\_flex\_input,site\_name,filename))

        # Read longitudes and latitudes from the dataset

        longitudes = temp\_ds['longitude'].values

        latitudes = temp\_ds['latitude'].values

        # Get the indices of the closest longitudes and latitudes to the site

        lon\_left = np.where(longitudes <= site\_lon)[0][-1]

        lon\_right = np.where(longitudes >= site\_lon)[0][0]

        lat\_top = np.where(latitudes >= site\_lat)[0][-1]

        lat\_bottom = np.where(latitudes <= site\_lat)[0][0]

        # print(f"lon\_left: {lon\_left}, lon\_right: {lon\_right}, lat\_top: {lat\_top}, lat\_bottom: {lat\_bottom}")

        # Now find the indices of the pixel where the site is located

        if abs(site\_lon - longitudes[lon\_left]) < abs(site\_lon - longitudes[lon\_right]):

            lon\_index = lon\_left

        else:

            lon\_index = lon\_right

        if abs(site\_lat - latitudes[lat\_top]) < abs(site\_lat - latitudes[lat\_bottom]):

            lat\_index = lat\_top

        else:

            lat\_index = lat\_bottom

        # Prepare empty lists to store results of each loop

        temp\_list\_sif\_name = []

        temp\_list\_sif\_avg = []

        # Get all variable names in the dataset

        temp\_name\_rar = list(temp\_ds.data\_vars)

        list\_indices = ['SIF\_FARRED\_max','SIF\_FARRED\_max\_wvl','SIF\_RED\_max','SIF\_RED\_max\_wvl','SIF\_O2B','SIF\_O2A','SIF\_int','SIF\_FARRED\_max\_un','SIF\_FARRED\_max\_wvl\_un','SIF\_RED\_max\_un','SIF\_RED\_max\_wvl\_un','SIF\_O2B\_un','SIF\_O2A\_un','SIF\_int\_un']

        # Loop through each variable name and calculate the average value in the ROI

        for var\_name in temp\_name\_rar:

            if var\_name in list\_indices:

                if roi == 300:

                    temp\_array = temp\_ds[var\_name][lat\_index,lon\_index].values

                elif roi == 600:

                    if abs(site\_lat - latitudes[lat\_index - 1]) < abs(site\_lat - latitudes[lat\_index + 1]):

                        lat\_index\_600 = lat\_index - 1

                    else:

                        lat\_index\_600 = lat\_index + 1

                    if abs(site\_lon - longitudes[lon\_index - 1]) < abs(site\_lon - longitudes[lon\_index + 1]):

                        lon\_index\_600 = lon\_index - 1

                    else:

                        lon\_index\_600 = lon\_index + 1

                    lat\_index\_start = min(lat\_index\_600, lat\_index)

                    lat\_index\_end = max(lat\_index\_600, lat\_index)

                    lon\_index\_start = min(lon\_index\_600, lon\_index)

                    lon\_index\_end = min(lon\_index\_600, lon\_index)

                    temp\_array = temp\_ds[var\_name][lat\_index\_start:(lat\_index\_end+1),lon\_index\_start:(lon\_index\_end+1)].values

                elif roi == 900:

                    temp\_array = temp\_ds[var\_name][(lat\_index-1):(lat\_index+2),(lon\_index-1):(lon\_index+2)].values

                else:

                    temp\_array = temp\_ds[var\_name][(lat\_index-1):(lat\_index+2),(lon\_index-1):(lon\_index+2)].values

                    print(f"The ROI {roi} m x {roi} m is greater than 900m x 900m. The code only supports 300m, 600m and 900m ROI. So now the calculation will be performed on a 900m ROI. ")

                temp\_list\_sif\_name.append(var\_name)

                temp\_avg = np.average(temp\_array).item()

                temp\_list\_sif\_avg.append(temp\_avg)

        # Output as a list

        list\_header = ['site\_code', 'latitude', 'longitude', 'flex\_date', 'flex\_time', 'flex\_filename', 's2\_filename'] + temp\_list\_sif\_name

        list\_value = [site\_name, site\_lat, site\_lon, filename.split('.')[0].split('\_')[-2], filename.split('.')[0].split('\_')[-1], filename, s2\_filename] + temp\_list\_sif\_avg

        # Export to the cache folder. This output csv file will be used for validation with FLOX data later.

        if not os.path.exists(os.path.join(self.path\_cache, 'FLEX', 'sif')):

            os.makedirs(os.path.join(self.path\_cache, 'FLEX', 'sif'))

        pd.DataFrame([list\_value], columns=list\_header).to\_csv(os.path.join(self.path\_cache, 'FLEX', 'sif',f'{site\_name}\_{filename}.csv'), index=False)

    def cal\_statistic\_flex\_flox(self) -> None:

        # Read matchup.csv

        df\_merge = pd.read\_csv(os.path.join(self.path\_output,"L2B\_FLEX\_FLOX\_matchup.csv"))

        num\_sites = df\_merge['site\_code'].nunique()

        num\_flex\_img = df\_merge['date'].nunique()

        #

        column\_pairs = [

            ['SIF\_FARRED\_max\_flox', 'SIF\_FARRED\_max\_flex'],

            ['SIF\_FARRED\_max\_wvl\_flox', 'SIF\_FARRED\_max\_wvl\_flex'],

            ['SIF\_RED\_max\_flox', 'SIF\_RED\_max\_flex'],

            ['SIF\_RED\_max\_wvl\_flox', 'SIF\_RED\_max\_wvl\_flex'],

            ['SIF\_O2B\_flox', 'SIF\_O2B\_flex'],

            ['SIF\_O2A\_flox', 'SIF\_O2A\_flex'],

            ['SIF\_int\_flox', 'SIF\_int\_flex'],

            ['SIF\_FARRED\_max\_un\_flox', 'SIF\_FARRED\_max\_un\_flex'],

            ['SIF\_FARRED\_max\_wvl\_un\_flox', 'SIF\_FARRED\_max\_wvl\_un\_flex'],

            ['SIF\_RED\_max\_un\_flox', 'SIF\_RED\_max\_un\_flex'],

            ['SIF\_RED\_max\_wvl\_un\_flox', 'SIF\_RED\_max\_wvl\_un\_flex'],

            ['SIF\_O2B\_un\_flox', 'SIF\_O2B\_un\_flex'],

            ['SIF\_O2A\_un\_flox', 'SIF\_O2A\_un\_flex'],

            ['SIF\_int\_un\_flox', 'SIF\_int\_un\_flex']]

        list\_num\_sites = []

        list\_num\_flex\_img = []

        list\_r\_2 = []

        list\_rmse = []

        list\_mean\_residual = []

        list\_random\_uncertainty = []

        list\_slope = []

        list\_intercept = []

        for pair in column\_pairs:

            list\_num\_sites.append(num\_sites)

            list\_num\_flex\_img.append(num\_flex\_img)

            # print(f"Calculating statistics for {pair[0]} and {pair[1]}...")

            temp\_r\_2, temp\_slope, temp\_intercept = self.cal\_r\_2(df\_merge[pair[0]],df\_merge[pair[1]])

            temp\_rmse = self.cal\_rmse(df\_merge[pair[0]],df\_merge[pair[1]])

            temp\_mean\_residual = self.cal\_mean\_residual(df\_merge[pair[0]],df\_merge[pair[1]])

            temp\_random\_uncertainty = self.cal\_random\_uncertainty(df\_merge[pair[0]],df\_merge[pair[1]],temp\_mean\_residual)

            list\_r\_2.append(temp\_r\_2)

            list\_rmse.append(temp\_rmse)

            list\_mean\_residual.append(temp\_mean\_residual)

            list\_random\_uncertainty.append(temp\_random\_uncertainty)

            list\_slope.append(temp\_slope)

            list\_intercept.append(temp\_intercept)

        df\_output = pd.DataFrame({

            "SIF metrics": ["SIF\_FARRED\_max","SIF\_FARRED\_max\_wvl","SIF\_RED\_max","SIF\_RED\_max\_wvl","SIF\_O2B","SIF\_O2A","SIF\_int","SIF\_FARRED\_max\_un","SIF\_FARRED\_max\_wvl\_un","SIF\_RED\_max\_un","SIF\_RED\_max\_wvl\_un","SIF\_O2B\_un","SIF\_O2A\_un","SIF\_int\_un"],

            'n\_sites': list\_num\_sites,

            'n\_images': list\_num\_flex\_img,

            'R^2': list\_r\_2,

            'RMSE': list\_rmse,

            'Bias': list\_mean\_residual,

            'Slope': list\_slope,

            'Intercept': list\_intercept,

            'Random uncertainty': list\_random\_uncertainty

        })

        df\_output.to\_csv(os.path.join(self.path\_output,"L2B\_validation\_report\_flex\_flox.csv"), index = False)

    def cal\_statistic\_flex\_tf(self) -> None:

        # Read matchup.csv

        df\_merge = pd.read\_csv(os.path.join(self.path\_output,"L2B\_FLEX\_FLOX\_matchup.csv"))

        # Remove empty rows

        df\_merge.replace('', np.nan, inplace=True)

        df\_merge.replace('N/A', np.nan, inplace=True)

        df\_merge.dropna(inplace = True)

        # Get number of sites and iamges

        num\_sites = df\_merge['site\_code'].nunique()

        num\_flex\_img = df\_merge['date'].nunique()

        #

        column\_pairs = [

            ['SIF\_FARRED\_max\_tf', 'SIF\_FARRED\_max\_flex'],

            ['SIF\_RED\_max\_tf', 'SIF\_RED\_max\_flex'],

            ['SIF\_O2B\_tf', 'SIF\_O2B\_flex'],

            ['SIF\_O2A\_tf', 'SIF\_O2A\_flex'],

            ['SIF\_int\_tf', 'SIF\_int\_flex'],

            ['SIF\_FARRED\_max\_un\_tf', 'SIF\_FARRED\_max\_un\_flex'],

            ['SIF\_RED\_max\_un\_tf', 'SIF\_RED\_max\_un\_flex'],

            ['SIF\_O2B\_un\_tf', 'SIF\_O2B\_un\_flex'],

            ['SIF\_O2A\_un\_tf', 'SIF\_O2A\_un\_flex'],

            ['SIF\_int\_un\_tf', 'SIF\_int\_un\_flex']]

        list\_num\_sites = []

        list\_num\_flex\_img = []

        list\_r\_2 = []

        list\_rmse = []

        list\_mean\_residual = []

        list\_random\_uncertainty = []

        list\_slope = []

        list\_intercept = []

        for pair in column\_pairs:

            list\_num\_sites.append(num\_sites)

            list\_num\_flex\_img.append(num\_flex\_img)

            # print(f"Calculating statistics for {pair[0]} and {pair[1]}...")

            temp\_r\_2, temp\_slope, temp\_intercept = self.cal\_r\_2(df\_merge[pair[0]],df\_merge[pair[1]])

            temp\_rmse = self.cal\_rmse(df\_merge[pair[0]],df\_merge[pair[1]])

            temp\_mean\_residual = self.cal\_mean\_residual(df\_merge[pair[0]],df\_merge[pair[1]])

            temp\_random\_uncertainty = self.cal\_random\_uncertainty(df\_merge[pair[0]],df\_merge[pair[1]],temp\_mean\_residual)

            list\_r\_2.append(temp\_r\_2)

            list\_rmse.append(temp\_rmse)

            list\_mean\_residual.append(temp\_mean\_residual)

            list\_random\_uncertainty.append(temp\_random\_uncertainty)

            list\_slope.append(temp\_slope)

            list\_intercept.append(temp\_intercept)

        df\_output = pd.DataFrame({

            "SIF metrics": ["SIF\_FARRED\_max","SIF\_RED\_max","SIF\_O2B","SIF\_O2A","SIF\_int","SIF\_FARRED\_max\_un","SIF\_RED\_max\_un","SIF\_O2B\_un","SIF\_O2A\_un","SIF\_int\_un"],

            'n\_sites': list\_num\_sites,

            'n\_images': list\_num\_flex\_img,

            'R^2': list\_r\_2,

            'RMSE': list\_rmse,

            'Bias': list\_mean\_residual,

            'Slope': list\_slope,

            'Intercept': list\_intercept,

            'Random uncertainty': list\_random\_uncertainty

        })

        df\_output.to\_csv(os.path.join(self.path\_output,"L2B\_validation\_report\_flex\_tf.csv"), index = False)

    def cal\_r\_2(self, x: np.array, y: np.array) -> float:

        # Fit linear model

        model = LinearRegression()

        model.fit(y.to\_numpy().reshape(-1,1), x.to\_numpy())

        # Predict using model

        y\_pred\_from\_model = model.predict(y.to\_numpy().reshape(-1,1))

        # Now compute R²

        r2 = r2\_score(x.to\_numpy(), y\_pred\_from\_model)

        # Get linear regression parameters

        slope = model.coef\_[0]      # coefficient (slope)

        intercept = model.intercept\_  # intercept

        return r2, slope, intercept

    def cal\_rmse(self, x: np.array, y: np.array) -> float:

        return np.sqrt(((x - y) \*\* 2).mean())

    def cal\_mean\_residual(self, x: np.array, y: np.array) -> float:

        return (x - y).mean()

    def cal\_random\_uncertainty(self, x: np.array, y: np.array, mean\_residual: float) -> float:

        return ((x - y - mean\_residual) \*\* 2).mean()

class S2(CalVal):

    def \_\_init\_\_(self, site\_name, site\_lat, site\_lon, s2\_l2a\_name):

        '''

        Args:

            site\_name (str): the name of the site.

            site\_lat (float): the latitude of the site.

            site\_lon (float): the longitude of the site.

            s2\_name (str): the name of the S2 L2A image, ending with ".SAFE".

        '''

        super().\_\_init\_\_()

        self.\_\_S2\_RESOLUTION = 10

        # Default threshold of CV

        self.\_threshold\_cv = 0.2

        # Default ROI

        self.\_area = 900

        # Default cloud coverage

        self.\_cloud = 0.5

        # Site name

        self.site\_name = site\_name

        self.site\_lat = site\_lat

        self.site\_lon = site\_lon

        # S2 L2A name

        self.s2\_l2a\_name = s2\_l2a\_name

        # S2 L2A images

        self.path\_l2a\_b04 = None

        self.path\_l2a\_b08 = None

        # S2 L2A MASK

        self.path\_l2a\_mask = None

        # S2 L2A MTD\_DS

        self.path\_l2a\_mtd\_ds = None

        # S2 L2A MTD\_TL

        self.path\_l2a\_mtd\_tl = None

        # S2 image CRS

        self.s2\_crs = None

        # S2 image quantification

        self.s2\_l2a\_quantification = None

        # S2 image offsets

        self.s2\_l2a\_offset\_b4 = None

        self.s2\_l2a\_offset\_b8 = None

        self.\_\_s2\_initialization()

    # ------------------------------ Private Members ----------------------------- #

    @property

    def s2\_resolution(self):

        return self.\_\_S2\_RESOLUTION

    @s2\_resolution.setter

    def s2\_resolution(self, value):

        if value:

            raise AttributeError("Cannot modify the spatial resolution of Sentinel-2 images!")

    @property

    def threshold\_cv(self):

        return self.\_threshold\_cv

    @threshold\_cv.setter

    def threshold\_cv(self, value):

        if not value:

            self.\_threshold\_cv = 0.2

        else:

            if value <= 0:

                raise ValueError("The threshold of CV should be greater than 0!!!")

            self.\_threshold\_cv = value

    @property

    def area(self):

        return self.\_area

    @area.setter

    def area(self, value):

        if not value:

            self.\_area = 900

        else:

            if value % self.\_\_S2\_RESOLUTION != 0:

                raise ValueError("The size of the ROI must be a multiple of 100 squared meters!!!")

            self.\_area = value

    @property

    def cloud(self):

        return self.\_cloud

    @cloud.setter

    def cloud(self, value):

        if not value:

            self.\_cloud = 0.5

        else:

            if value < 0 or value > 1:

                raise ValueError("The cloud coverage must be between 0 and 1!!!")

            self.\_cloud = value

    # ------------------------------ Private Methods ------------------------------ #

    def \_\_s2\_initialization(self) -> None:

        '''

        Get all necessary data of the current S2 image.

        '''

        self.path\_l2a\_b04, self.path\_l2a\_b08, self.path\_l2a\_mask, self.path\_l2a\_mtd\_ds, self.path\_l2a\_mtd\_tl = self.get\_s2\_l2a\_paths()

        self.s2\_crs = self.get\_s2\_crs()

        self.quantification\_l2a, self.offset\_l2a\_b04, self.offset\_l2a\_b08 = self.get\_s2\_l2a\_metadata()

    # ------------------------------ Public Methods ------------------------------ #

    def create\_cache\_subfolder(self, subpath) -> None:

        '''

        Create subfolder inside the cache folder.

        Args:

            subpath (str): the part of the subfolder path (after 'cache').

        '''

        temp = os.path.join(self.path\_cache, subpath)

        if not os.path.exists(temp):

            os.makedirs(temp)

    def get\_s2\_l2a\_paths(self) -> tuple:

        '''

        Get the paths to S2 B4, B8, MSK\_CLASSI\_B00, MTD\_DS and MTD\_TL files.

        Returns:

            tuple: (path\_l2a\_b04, path\_l2a\_b08, path\_l2a\_mask, path\_l2a\_xml\_ds, path\_l2a\_xml\_tl)

        '''

        temp\_path\_l2a = os.path.join(self.path\_s2\_input, self.site\_name, self.s2\_l2a\_name)

        for path, subdirs, files in os.walk(temp\_path\_l2a):

            for name in files:

                temp = os.path.join(path, name)

                if temp[-3:] == 'jp2'in temp and "10m" in temp and "B04" in temp :

                    path\_l2a\_b04 = temp

                if temp[-3:] == 'jp2'in temp and "10m" in temp and "B08" in temp :

                    path\_l2a\_b08 = temp

                # Path to the mask file

                if "MSK\_CLASSI\_B00" in temp and temp[-3:] == 'jp2':

                    path\_l2a\_mask = temp

                # Get the path to the XML file "MTD\_DS" of L1C raster, where there are values of "Quantification Value" and "Radiometric Offset"

                if "MTD\_DS.xml" in temp:

                    path\_l2a\_xml\_ds = temp

                if "MTD\_TL.xml" in temp:

                    path\_l2a\_xml\_tl = temp

        if not os.path.exists(temp\_path\_l2a):

            print("User Error: Please organise the input S2 images in correct folder structure. ")

            raise FileNotFoundError(f"The input S2 images folder {temp\_path\_l2a} doesn't contain the correct folder structure or doesn't contain S2 images! Please check the input S2 images folder!")

        else:

            return path\_l2a\_b04, path\_l2a\_b08, path\_l2a\_mask, path\_l2a\_xml\_ds, path\_l2a\_xml\_tl

    def get\_s2\_crs(self) -> str:

        '''

        Get the coordinate reference system (CRS) of the S2 image.

        Returns:

            str: The CRS of the S2 image in EPSG format.

        '''

        # Read the DS xml file of L2A

        with open(self.path\_l2a\_mtd\_tl, 'r') as f:

            data = f.read()

        bs\_l2a\_tl = BeautifulSoup(data, "xml")

        # Get the quantification value!

        l2a\_crs = str(bs\_l2a\_tl.find("HORIZONTAL\_CS\_CODE").text)

        return l2a\_crs

    def get\_s2\_l2a\_metadata(self) -> tuple:

        '''

        Retrieve quantification, offset\_b04 and offset\_b08 from the L2A metadata MTD\_DS.xml file.

        Returns:

            tuple: (quantification\_l2a, offset\_l2a\_b04, offset\_l2a\_b08)

        '''

        # Read the DS xml file of L2A

        with open(self.path\_l2a\_mtd\_ds, 'r') as f:

            data = f.read()

        bs\_l2a\_ds = BeautifulSoup(data, "xml")

        # Get the quantification value!

        quantification\_l2a = int(bs\_l2a\_ds.find("BOA\_QUANTIFICATION\_VALUE").text)

        # Get the radiometric offset!

        offset\_l2a\_b04 = int(bs\_l2a\_ds.find("BOA\_ADD\_OFFSET", {"band\_id": "3"}).text)

        offset\_l2a\_b08 = int(bs\_l2a\_ds.find("BOA\_ADD\_OFFSET", {"band\_id": "7"}).text)

        return quantification\_l2a, offset\_l2a\_b04, offset\_l2a\_b08

    def create\_clipping\_shapefile(self) -> gpd.GeoDataFrame:

        '''

        Create a shapefile to be used for S2 image clipping. This shapefile overlapps perfectly with the pixels of the S2 images.

        Returns:

            gpd.GeoDataFrame: the new shapefile that will used to clip the S2 image.

        '''

        # Open the FLEX image

        temp\_ds = xr.open\_dataset(os.path.join(self.path\_flex\_input,self.site\_name,self.flex\_filename))

        # Read longitudes and latitudes from the dataset

        longitudes = temp\_ds['longitude'].values

        latitudes = temp\_ds['latitude'].values

        # Get the indices of the closest longitudes and latitudes to the site

        lon\_left = np.where(longitudes <= self.site\_lon)[0][-1]

        lon\_right = np.where(longitudes >= self.site\_lon)[0][0]

        lat\_top = np.where(latitudes >= self.site\_lat)[0][-1]

        lat\_bottom = np.where(latitudes <= self.site\_lat)[0][0]

        # print(f"lon\_left: {lon\_left}, lon\_right: {lon\_right}, lat\_top: {lat\_top}, lat\_bottom: {lat\_bottom}")

        # Now find the indices of the pixel where the site is located

        if abs(self.site\_lon - longitudes[lon\_left]) < abs(self.site\_lon - longitudes[lon\_right]):

            lon\_index = lon\_left

        else:

            lon\_index = lon\_right

        if abs(self.site\_lat - latitudes[lat\_top]) < abs(self.site\_lat - latitudes[lat\_bottom]):

            lat\_index = lat\_top

        else:

            lat\_index = lat\_bottom

        lon\_dif = abs(longitudes[1] - longitudes[0]) / 2

        lat\_dif = abs(latitudes[1] - latitudes[0]) / 2

        # Create a box geometry

        if self.area == 300:

            miny = latitudes[lat\_index] - lat\_dif

            maxy = latitudes[lat\_index] + lat\_dif

            minx = longitudes[lon\_index] - lon\_dif

            maxx = longitudes[lon\_index] + lon\_dif

        elif self.area == 600:

            if abs(self.site\_lat - latitudes[lat\_index - 1]) < abs(self.site\_lat - latitudes[lat\_index + 1]):

                lat\_index\_600 = lat\_index - 1

            else:

                lat\_index\_600 = lat\_index + 1

            if abs(self.site\_lon - longitudes[lon\_index - 1]) < abs(self.site\_lon - longitudes[lon\_index + 1]):

                lon\_index\_600 = lon\_index - 1

            else:

                lon\_index\_600 = lon\_index + 1

            miny = min(lat\_index\_600, lat\_index) - lat\_dif

            maxy = max(lat\_index\_600, lat\_index) + lat\_dif

            minx = min(lon\_index\_600, lon\_index) - lon\_dif

            maxx = min(lon\_index\_600, lon\_index) + lon\_dif

        else:

            miny = min(latitudes[lat\_index - 1], latitudes[lat\_index + 1]) - lat\_dif

            maxy = max(latitudes[lat\_index - 1], latitudes[lat\_index + 1]) + lat\_dif

            minx = min(longitudes[lon\_index - 1],longitudes[lon\_index + 1]) - lon\_dif

            maxx = max(longitudes[lon\_index - 1],longitudes[lon\_index + 1]) + lon\_dif

        geom = shp.geometry.box(minx, miny, maxx, maxy)

        gdf\_new = gpd.GeoDataFrame({'value': [0], 'geometry': geom}, crs="EPSG:4326")

        gdf\_new\_utm = gdf\_new.to\_crs(self.s2\_crs)

        # Export shapefiles

        gdf\_new.to\_file(os.path.join(self.path\_cache,self.site\_name,"roi\_4326.shp"))

        gdf\_new\_utm.to\_file(os.path.join(self.path\_cache,self.site\_name,"roi\_utm.shp"))

        return gdf\_new\_utm

    def create\_clipping\_raster(self, list\_indices = ['NDVI','NIRvREF','TF2']) -> None:

        '''

        '''

        # Suppress divide by zero warning

        np.seterr(all='ignore')

        # Read values

        img\_l2a\_b04 = rio.open(self.path\_l2a\_b04)

        img\_l2a\_b08 = rio.open(self.path\_l2a\_b08)

        values\_l2a\_b04 = img\_l2a\_b04.read(1).astype(np.int32)

        values\_l2a\_b08 = img\_l2a\_b08.read(1).astype(np.int32)

        # Get metadata

        src = img\_l2a\_b04

        out\_meta = src.meta

        out\_meta.update({

            "driver": "GTiff",

            "dtype": "float64",

            "crs": src.crs,

            "transform": src.transform

        })

        # ------------------------------- Read Mask ROI ------------------------------ #

        img\_mask = rio.open(os.path.join(self.path\_cache,self.site\_name,"Mask.tif"))

        values\_mask = img\_mask.read(1)

        values\_mask = np.where(values\_mask == 0, 1, np.nan)

        # ----------------------------------- NDVI ----------------------------------- #

        if not os.path.exists(os.path.join(self.path\_cache, self.site\_name)):

            os.makedirs(os.path.join(self.path\_cache, self.site\_name))

        # Calculate NDVI of L2A!

        # NDVI = (B8 - B4) / (B8 + B4)

        temp\_ndvi = ((values\_l2a\_b08 + self.offset\_l2a\_b08).astype(float) / self.quantification\_l2a - (values\_l2a\_b04 + self.offset\_l2a\_b04).astype(float) / self.quantification\_l2a) / ((values\_l2a\_b08 + self.offset\_l2a\_b08).astype(float) / self.quantification\_l2a + (values\_l2a\_b04 + self.offset\_l2a\_b04).astype(float) / self.quantification\_l2a )

        temp\_ndvi = temp\_ndvi \* values\_mask

        if 'NDVI' in list\_indices:

            # Save

            with rio.open(os.path.join(self.path\_cache, self.site\_name, "NDVI.tif"), 'w',\*\*out\_meta) as dest:

                dest.write(temp\_ndvi, 1)

            # Clip to the ROI!

            self.clip\_raster\_by\_shapefile(os.path.join(self.path\_cache, self.site\_name, "NDVI.tif"))

        # ---------------------------------- NIRvREF --------------------------------- #

        if 'NIRvREF' in list\_indices:

            # Calculate NIRvREF of L2A!

            # NIRvREF = NDVI \* B8

            temp\_nirvref = temp\_ndvi \* (values\_l2a\_b08 + self.offset\_l2a\_b08).astype(float) / self.quantification\_l2a

            temp\_nirvref = temp\_nirvref \* values\_mask

            # Save

            with rio.open(os.path.join(self.path\_cache, self.site\_name, "NIRv.tif"), 'w',\*\*out\_meta) as dest:

                dest.write(temp\_nirvref, 1)

            # Clip to the ROI!

            self.clip\_raster\_by\_shapefile(os.path.join(self.path\_cache, self.site\_name, "NIRv.tif"))

        # ------------------------------------ TF2 ----------------------------------- #

        if 'TF2' in list\_indices:

            # Calculate transfer function 2! B4 \* NIRvREF ^ 2

            temp\_tf2 = (values\_l2a\_b04 + self.offset\_l2a\_b04).astype(float) / self.quantification\_l2a \* (temp\_nirvref \*\* 2)

            temp\_tf2 = temp\_tf2 \* values\_mask

            # Save

            with rio.open(os.path.join(self.path\_cache, self.site\_name, "TF2.tif"), 'w',\*\*out\_meta) as dest:

                dest.write(temp\_tf2, 1)

            # Clip to the ROI!

            self.clip\_raster\_by\_shapefile(os.path.join(self.path\_cache, self.site\_name, "TF2.tif"))

    def cal\_l2a\_indices(self) -> dict:

        '''

        Calculate the NDVI and NIRVref of L2A images using the values of B04 and B08 bands.

        Returns:

            dict: A dict containing the NDVI, NIRVref and transfer function 2 values of L2A images.

        '''

        # Suppress divide by zero warning

        np.seterr(all='ignore')

        if not os.path.exists(os.path.join(self.path\_cache, self.site\_name, "NDVI\_ROI.tif")) or not os.path.exists(os.path.join(self.path\_cache, self.site\_name, "NIRvREF\_ROI.tif")):

            self.create\_clipping\_raster(['NDVI','NIRvREF'])

        # Calculate avg, std, cv of NDVI inside the ROI

        image\_ndvi\_roi = rio.open(os.path.join(self.path\_cache, self.site\_name, "NDVI\_ROI.tif"))

        values\_ndvi\_roi = image\_ndvi\_roi.read(1)

        temp\_ndvi\_std = self.cal\_std(values\_ndvi\_roi)

        temp\_ndvi\_avg = self.cal\_avg(values\_ndvi\_roi)

        temp\_ndvi\_cv = self.cal\_cv(values\_ndvi\_roi)

        temp\_ndvi\_flag = self.cal\_flag(temp\_ndvi\_cv)

        # Calculate avg, std, cv of NIRvREF inside the ROI

        image\_nirv\_roi = rio.open(os.path.join(self.path\_cache, self.site\_name, "NIRv\_ROI.tif"))

        values\_nirv\_roi = image\_nirv\_roi.read(1)

        temp\_nirv\_std = self.cal\_std(values\_nirv\_roi)

        temp\_nirv\_avg = self.cal\_avg(values\_nirv\_roi)

        temp\_nirv\_cv = self.cal\_cv(values\_nirv\_roi)

        temp\_nirv\_flag = self.cal\_flag(temp\_nirv\_cv)

        # Manually ensure available memory

        image\_nirv\_roi.close()

        image\_ndvi\_roi.close()

        return temp\_ndvi\_std, temp\_ndvi\_avg, temp\_ndvi\_cv, temp\_ndvi\_flag, temp\_nirv\_std, temp\_nirv\_avg, temp\_nirv\_cv, temp\_nirv\_flag

    def clip\_raster\_by\_shapefile(self, path\_raster) -> None:

        '''

        Clip the raster to the shapefile and save to local storage.

        Args:

            path\_raster (str): path to the raster to be clipped.

        '''

        # Create the clipping shapefile

        shp\_clipping = self.create\_clipping\_shapefile()

        # Read the raster to be clipped

        raster = rio.open(path\_raster)

        # Clipping!

        out\_image, out\_transform = rio.mask.mask(raster, shp\_clipping.geometry, crop=True)

        out\_meta = raster.meta

        out\_meta.update({"driver": "GTiff",

                        "height": out\_image.shape[1],

                        "width": out\_image.shape[2],

                        "transform": out\_transform})

        # Save!

        with rio.open(os.path.join(self.path\_cache, self.site\_name, os.path.splitext(os.path.basename(path\_raster))[0] + "\_ROI.tif"), "w", \*\*out\_meta) as dest:

            dest.write(out\_image)

        # Manually ensure available memory

        raster.close()

    def cal\_valid\_pixels(self) -> tuple:

        '''

        Check if there are sufficient valid pixels (not snow, ice or cloud).

        Returns:

            tuple: (bool\_pass, num\_valid\_pixels, percentage\_valid\_pixels)

        '''

        mask\_l2a = rio.open(self.path\_l2a\_mask)

        mask\_l2a\_opaque\_clouds = mask\_l2a.read(1)

        mask\_l2a\_cirrus\_clouds = mask\_l2a.read(2)

        mask\_l2a\_snowice\_areas = mask\_l2a.read(3)

        # Check if all three masks are empty. If not empty, we should check if the masked

        mask\_combined = mask\_l2a\_opaque\_clouds + mask\_l2a\_cirrus\_clouds + mask\_l2a\_snowice\_areas

        if np.max(mask\_combined) >= 1:

            # Upscale 60mx60m mask to 10mx10m without modifying any pixel values

            mask\_combined\_upscale = np.repeat(mask\_combined, 6, axis = 0)

            mask\_combined\_upscale = np.repeat(mask\_combined\_upscale, 6, axis = 1)

            # Read a random S2 image to retrieve metadata

            img\_l2a\_b04 = rio.open(self.path\_l2a\_b04)

            mask\_meta = img\_l2a\_b04.meta

            # Save this mask to the cache folder

            with rio.open(os.path.join(self.path\_cache,self.site\_name,"Mask.tif"), "w", \*\*mask\_meta) as dest:

                dest.write(mask\_combined\_upscale, indexes = 1)

            # clip and save the mask raster

            self.clip\_raster\_by\_shapefile(os.path.join(self.path\_cache, self.site\_name, "Mask.tif"))

            # Validate pixels in the ROI

            temp\_mask\_clipped = rio.open(os.path.join(self.path\_cache, self.site\_name, "Mask\_ROI.tif"))

            temp\_mask\_clipped\_values = temp\_mask\_clipped.read(1)

            # Manually ensure available memory

            mask\_l2a.close()

            img\_l2a\_b04.close()

            temp\_mask\_clipped.close()

            if np.max(temp\_mask\_clipped\_values) >= 1:

                temp\_valid\_pixels = np.count\_nonzero(temp\_mask\_clipped\_values == 0)

                temp\_invalid\_pixels = np.count\_nonzero(temp\_mask\_clipped\_values != 0)

                temp\_total\_pixels = temp\_valid\_pixels + temp\_invalid\_pixels

                temp\_valid\_pixels\_ratio = temp\_valid\_pixels / temp\_total\_pixels

                if temp\_valid\_pixels\_ratio >= self.cloud:

                    print(f"But the ratio of valid pixels is {temp\_valid\_pixels\_ratio:.2%}, equal to or greater than {self.cloud:.2%}, so we can use these S2 images. ")

                    bool\_pass = True

                    return bool\_pass, temp\_valid\_pixels, temp\_valid\_pixels\_ratio

                else:

                    print(f"And the ratio of valid pixels is {temp\_valid\_pixels\_ratio:.2%}, lower than {self.cloud:.2%}, so we can't use these S2 images and hence we can't proceed. ")

                    bool\_pass = False

                    return bool\_pass, temp\_valid\_pixels, temp\_valid\_pixels\_ratio

            else:

                print(f"All pixels in the current S2 image are valid! ")

                bool\_pass = True

                return bool\_pass, (self.area / 10) \*\* 2, 1

        else:

            print(f"All pixels in the current S2 image are valid! ")

            bool\_pass = True

        # Manually ensure available memory

        mask\_l2a.close()

        return bool\_pass, (self.area / 10) \*\* 2, 1

    def cal\_std(self, value):

        return np.nanstd(value)

    def cal\_avg(self, value):

        return np.nanmean(value)

    def cal\_cv(self, value):

        return np.nanstd(value) / np.nanmean(value)

    def cal\_flag(self, value):

        if value <= self.threshold\_cv:

            return 1

        else:

            return 0

    def cal\_transfer\_function(self, flex\_date) -> bool:

        '''

        Application of transfer function, and then save the calculated averages into a temporary .csv file in "Cache\\TF

        Args:

            flex\_date (int): The date of the current flex image.

        Returns:

            bool: The validality of FLOX

        '''

        if not os.path.exists(os.path.join(self.path\_cache, self.site\_name, "NIRv\_ROI.tif")) or not os.path.exists(os.path.join(self.path\_cache, self.site\_name, "TF2\_ROI.tif")):

            self.create\_clipping\_raster(['NIRvREF','TF2'])

        # ------------------------ Find the index of the site ------------------------ #

        # Create a point shapefile of the site, using Lon-Lat

        df\_4326 = pd.DataFrame({

            "Site": [self.site\_name],

            "Latitude": [self.site\_lat],

            "Longitude": [self.site\_lon]

        })

        gdf\_4326 = gpd.GeoDataFrame(

            df\_4326,

            geometry = gpd.points\_from\_xy(df\_4326['Longitude'], df\_4326['Latitude']),

            crs = "EPSG:4326"

        )

        # Convert the crs from lat-lon to that of the S2 image

        gdf\_s2\_crs = gdf\_4326.to\_crs(self.s2\_crs)

        # Retrieve the coordinates in the new crs of our site

        site\_x = gdf\_s2\_crs.geometry.x.values[0]

        site\_y = gdf\_s2\_crs.geometry.y.values[0]

        # Open image

        img\_tf1 = rio.open(os.path.join(self.path\_cache, self.site\_name, "NIRv\_ROI.tif"))

        img\_tf2 = rio.open(os.path.join(self.path\_cache, self.site\_name, "TF2\_ROI.tif"))

        # Read values

        value\_tf1 = img\_tf1.read(1)

        value\_tf2 = img\_tf2.read(1)

        # Get the corresponding FLOX data

        df\_flox = pd.read\_csv(self.file\_flox\_csv, sep = ';')

        df\_flox.rename(columns={'UTC\_datetime': 'date'}, inplace=True)

        df\_flox['date'] = pd.to\_datetime(df\_flox['date'], format='%d/%m/%Y %H:%M')

        # Convert to string date format

        df\_flox['date'] = df\_flox['date'].dt.strftime('%Y%m%d').astype(str)

        df\_flox\_site = df\_flox[(df\_flox['ID\_SITE'] == self.site\_name) & (df\_flox['date'] == flex\_date)]

        # Create an empty dict

        temp\_dict = {'site\_code': self.site\_name, 'date': str(flex\_date),

            'SIF\_O2A': 0, 'SIF\_FARRED\_max': 0, 'SIF\_int': 0, 'SIF\_O2B': 0, 'SIF\_RED\_max': 0,

            'SIF\_O2A\_un': 0, 'SIF\_FARRED\_max\_un': 0, 'SIF\_int\_un': 0, 'SIF\_O2B\_un': 0, 'SIF\_RED\_max\_un': 0,

        }

        # ------------------------------------ TF1 ----------------------------------- #

        for var\_name in ['SIF\_O2A','SIF\_FARRED\_max','SIF\_int']:

            # Get the pixel index of the site

            site\_row, site\_col = img\_tf1.index(site\_x, site\_y)

            # Get the value of the site based on the transfer function

            value\_s2\_flox = value\_tf1[site\_row, site\_col]

            # Get the value of the flox of the current index

            value\_flox = df\_flox\_site[var\_name].values[0].item()

            if value\_s2\_flox == np.nan or not value\_s2\_flox:

                print(f"{self.site\_name} is inside an invalid pixel. The transfer function won't be applied!")

                temp\_dict[var\_name] = 'N/A'

                bool\_flox\_invalid = True

            else:

                # Apply transfer function 1

                value\_tf = value\_tf1 / value\_s2\_flox \* value\_flox

                # Calculate average

                value\_tf\_avg = np.nanmean(value\_tf)

                # Update the dicct

                temp\_dict[var\_name] = value\_tf\_avg

                bool\_flox\_invalid = False

        # ------------------------------------ TF2 ----------------------------------- #

        for var\_name in ['SIF\_O2B','SIF\_RED\_max']:

            # Get the pixel index of the site

            site\_row, site\_col = img\_tf2.index(site\_x, site\_y)

            # Get the value of the site based on the transfer function

            value\_s2\_flox = value\_tf2[site\_row, site\_col]

            # Get the value of the flox of the current index

            value\_flox = df\_flox\_site[var\_name].values[0].item()

            if value\_s2\_flox == np.nan or not value\_s2\_flox:

                print(f"{self.site\_name} is inside an invalid pixel. The transfer function won't be applied!")

                temp\_dict[var\_name] = 'N/A'

            else:

                # Apply transfer function 2

                value\_tf = value\_tf2 / value\_s2\_flox \* value\_flox

                # Calculate average

                value\_tf\_avg = np.nanmean(value\_tf)

                # Update the dicct

                temp\_dict[var\_name] = value\_tf\_avg

        # Save to local storage

        df\_dict = pd.DataFrame([temp\_dict])

        if not os.path.exists(os.path.join(self.path\_cache,"TF")):

            os.makedirs(os.path.join(self.path\_cache,"TF"))

        df\_dict.to\_csv(os.path.join(self.path\_cache,"TF",self.site\_name + "\_" + flex\_date + ".csv"), index = False)

        return bool\_flox\_invalid

    def remove\_cache(self):

        # Delete cache folder?

        if self.bool\_delete\_cache:

            if os.path.exists(os.path.join(self.path\_cache, self.site\_name, "NDVI.tif")):

                os.remove(os.path.join(self.path\_cache, self.site\_name, "NDVI.tif"))

            if os.path.exists(os.path.join(self.path\_cache, self.site\_name, "NIRv.tif")):

                os.remove(os.path.join(self.path\_cache, self.site\_name, "NIRv.tif"))

            if os.path.exists(os.path.join(self.path\_cache, self.site\_name, "TF2.tif")):

                os.remove(os.path.join(self.path\_cache, self.site\_name, "TF2.tif"))

            if os.path.exists(os.path.join(self.path\_cache, self.site\_name, "NDVI\_ROI.tif")):

                os.remove(os.path.join(self.path\_cache, self.site\_name, "NDVI\_ROI.tif"))

            if os.path.exists(os.path.join(self.path\_cache, self.site\_name, "NIRv\_ROI.tif")):

                os.remove(os.path.join(self.path\_cache, self.site\_name, "NIRv\_ROI.tif"))

            if os.path.exists(os.path.join(self.path\_cache, self.site\_name, "TF2\_ROI.tif")):

                os.remove(os.path.join(self.path\_cache, self.site\_name, "TF2\_ROI.tif"))

## TDS and Auxiliar Files

**Example of installation file:**

**requirements.txt:**

Numpy >=1.26.4

pandas >=2.2.2

shapely >=2.0.5

geopandas >=1.0.1

matplotlib >=3.9.1

lxml >=5.2.2

beautifulsoup4 >=4.12.3

GDAL ==3.9.1

rasterio ==1.3.10

configparser >=7.1.0

xarray >=2024.10.0

scipy >=1.14.1

netcdf4 >=1.7.2

**environment.yml**

name: cal-val-prototype

channels:

- conda-forge

- defaults

dependencies:

- python =3.11

- numpy >=1.26.4

- pandas >=2.2.2

- shapely >=2.0.5

- geopandas >=1.0.1

- lxml >=5.2.2

- beautifulsoup4 >=4.12.3

- gdal ==3.9.1

- rasterio ==1.3.10

- configparser >=7.1.0

- xarray >=2024.10.0

- scipy >=1.14.1

- netcdf4 >=1.7.2

**Input files:**

**Site.csv:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| site\_code | latitude | longitude | reference\_area(m) | time\_window(days) | threshold\_cloud(%) | threshold\_CV(%) | vegetation\_pixel(%) |
| IT-CP2 | 41.7043 | 12.3573 | 900 | 15 | 50 | 20 | 50 |
| IT-JDS | 44.87431 | 11.9792 | 900 | 15 | 50 | 20 | 50 |
| US-LIN | 41.1797 | -96.4404 | 900 | 15 | 50 | 20 | 50 |
| IT-SR2 | 43.732 | 10.291 | 900 | 20 | 50 | 20 | 50 |
| IT-JDS2 | 44.85315 | 12.06556 | 900 | 20 | 50 | 20 | 50 |

**Input S2 Images\:**

…

**Input FLEX Images\**

**…**

**Output file:**

Full\_Spectrum\_avg\_FLEX\_table.csv

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| site\_code | latitude | longitude | flex\_date | flex\_time | flex\_filename | s2\_filename | Sif Emission Spectrum\_sif\_wavelength\_grid=670 | Sif Emission Spectrum\_sif\_wavelength\_grid=672 | Sif Emission Spectrum\_sif\_wavelength\_grid=674 | Sif Emission Spectrum\_sif\_wavelength\_grid=676 | Sif Emission Spectrum\_sif\_wavelength\_grid=678 | Sif Emission Spectrum\_sif\_wavelength\_grid=680 | Sif Emission Spectrum\_sif\_wavelength\_grid=682 | Sif Emission Spectrum\_sif\_wavelength\_grid=684 | Sif Emission Spectrum\_sif\_wavelength\_grid=686 | Sif Emission Spectrum\_sif\_wavelength\_grid=688 | Sif Emission Spectrum\_sif\_wavelength\_grid=690 | Sif Emission Spectrum\_sif\_wavelength\_grid=692 | Sif Emission Spectrum\_sif\_wavelength\_grid=694 | Sif Emission Spectrum\_sif\_wavelength\_grid=696 | Sif Emission Spectrum\_sif\_wavelength\_grid=698 | Sif Emission Spectrum\_sif\_wavelength\_grid=700 | Sif Emission Spectrum\_sif\_wavelength\_grid=702 | Sif Emission Spectrum\_sif\_wavelength\_grid=704 | Sif Emission Spectrum\_sif\_wavelength\_grid=706 | Sif Emission Spectrum\_sif\_wavelength\_grid=708 | Sif Emission Spectrum\_sif\_wavelength\_grid=710 | Sif Emission Spectrum\_sif\_wavelength\_grid=712 | Sif Emission Spectrum\_sif\_wavelength\_grid=714 | Sif Emission Spectrum\_sif\_wavelength\_grid=716 | Sif Emission Spectrum\_sif\_wavelength\_grid=718 | Sif Emission Spectrum\_sif\_wavelength\_grid=720 | Sif Emission Spectrum\_sif\_wavelength\_grid=722 | Sif Emission Spectrum\_sif\_wavelength\_grid=724 | Sif Emission Spectrum\_sif\_wavelength\_grid=726 | Sif Emission Spectrum\_sif\_wavelength\_grid=728 | Sif Emission Spectrum\_sif\_wavelength\_grid=730 | Sif Emission Spectrum\_sif\_wavelength\_grid=732 | Sif Emission Spectrum\_sif\_wavelength\_grid=734 | Sif Emission Spectrum\_sif\_wavelength\_grid=736 | Sif Emission Spectrum\_sif\_wavelength\_grid=738 | Sif Emission Spectrum\_sif\_wavelength\_grid=740 | Sif Emission Spectrum\_sif\_wavelength\_grid=742 | Sif Emission Spectrum\_sif\_wavelength\_grid=744 | Sif Emission Spectrum\_sif\_wavelength\_grid=746 | Sif Emission Spectrum\_sif\_wavelength\_grid=748 | Sif Emission Spectrum\_sif\_wavelength\_grid=750 | Sif Emission Spectrum\_sif\_wavelength\_grid=752 | Sif Emission Spectrum\_sif\_wavelength\_grid=754 | Sif Emission Spectrum\_sif\_wavelength\_grid=756 | Sif Emission Spectrum\_sif\_wavelength\_grid=758 | Sif Emission Spectrum\_sif\_wavelength\_grid=760 | Sif Emission Spectrum\_sif\_wavelength\_grid=762 | Sif Emission Spectrum\_sif\_wavelength\_grid=764 | Sif Emission Spectrum\_sif\_wavelength\_grid=766 | Sif Emission Spectrum\_sif\_wavelength\_grid=768 | Sif Emission Spectrum\_sif\_wavelength\_grid=770 | Sif Emission Spectrum\_sif\_wavelength\_grid=772 | Sif Emission Spectrum\_sif\_wavelength\_grid=774 | Sif Emission Spectrum\_sif\_wavelength\_grid=776 | Sif Emission Spectrum\_sif\_wavelength\_grid=778 | Sif Emission Spectrum\_sif\_wavelength\_grid=780 |
| IT-JDS2 | 44.85314969 | 12.06555824 | 20220717 | 93225 | PRS\_TD\_20220717\_093225.nc | S2A\_MSIL2A\_20220717T100611\_N0510\_R022\_T32TQQ\_20240717T130024.SAFE | 0.047608148 | 0.055653043 | 0.06467282 | 0.07637988 | 0.092011899 | 0.109093539 | 0.125056162 | 0.139806569 | 0.153429464 | 0.163856462 | 0.169714227 | 0.173986763 | 0.180047974 | 0.188398063 | 0.198655516 | 0.210676223 | 0.224307641 | 0.239447966 | 0.256207466 | 0.275806814 | 0.299415946 | 0.326712608 | 0.357027173 | 0.389817119 | 0.424802661 | 0.4628824 | 0.504514098 | 0.547213912 | 0.588726997 | 0.630790949 | 0.675101519 | 0.718420148 | 0.757390022 | 0.791934788 | 0.821774602 | 0.842755795 | 0.850775242 | 0.84623152 | 0.8302688 | 0.802929699 | 0.765028059 | 0.722514689 | 0.68036586 | 0.635843456 | 0.58619231 | 0.536770046 | 0.493349612 | 0.454651028 | 0.417681813 | 0.380771935 | 0.343404263 | 0.308562875 | 0.279087514 | 0.254308581 | 0.232912034 | 0.214423165 |
| IT-JDS | 44.874305 | 11.979201 | 20220717 | 93225 | PRS\_TD\_20220717\_093225.nc | S2A\_MSIL2A\_20220717T100611\_N0510\_R022\_T32TQQ\_20240717T130024.SAFE | 0.05042259 | 0.058984596 | 0.068546921 | 0.081042357 | 0.097657748 | 0.115748152 | 0.132533088 | 0.147910774 | 0.161932141 | 0.172460854 | 0.17803888 | 0.181819737 | 0.187448606 | 0.195366338 | 0.205267757 | 0.216847658 | 0.230072126 | 0.244737759 | 0.261014044 | 0.280070871 | 0.303078055 | 0.329645753 | 0.359129548 | 0.390800506 | 0.424489051 | 0.461025923 | 0.500851929 | 0.541491866 | 0.580724478 | 0.620313168 | 0.661868513 | 0.702406704 | 0.738587499 | 0.770377159 | 0.797656894 | 0.81637466 | 0.822698593 | 0.817001998 | 0.800573111 | 0.773310244 | 0.73616302 | 0.694692373 | 0.653577864 | 0.61029309 | 0.562176347 | 0.514398634 | 0.472470164 | 0.435068637 | 0.399430305 | 0.36394465 | 0.328068972 | 0.294668645 | 0.266439468 | 0.242703229 | 0.222222224 | 0.204623491 |
| IT-SR2 | 43.732 | 10.291 | 20230616 | 101431 | PRS\_TD\_20230616\_101431.nc | S2B\_MSIL2A\_20230617T100559\_N0509\_R022\_T32TPP\_20230617T131301.SAFE | 0.249518931 | 0.295007795 | 0.346159399 | 0.412468314 | 0.500394166 | 0.596119165 | 0.683587313 | 0.759271979 | 0.820341945 | 0.854454279 | 0.853623569 | 0.83485496 | 0.817866623 | 0.806337595 | 0.799555779 | 0.797809482 | 0.801403821 | 0.810135365 | 0.824343264 | 0.846773982 | 0.879513085 | 0.920407295 | 0.966286123 | 1.015199661 | 1.066198707 | 1.121249795 | 1.181268692 | 1.240795851 | 1.29515183 | 1.348727942 | 1.405678153 | 1.460203648 | 1.50577724 | 1.543094039 | 1.572849035 | 1.587887645 | 1.581326365 | 1.554487467 | 1.510202289 | 1.448454261 | 1.370819688 | 1.286691546 | 1.204208136 | 1.118570924 | 1.025118113 | 0.933360398 | 0.85319972 | 0.782177508 | 0.715190351 | 0.649118781 | 0.583199739 | 0.522299349 | 0.471096843 | 0.428422451 | 0.391868651 | 0.360621572 |
| US-LIN | 41.1797 | -96.44039 | 20230711 | 171357 | PRS\_TD\_20230711\_171357.nc | S2A\_MSIL2A\_20230711T171901\_N0510\_R012\_T14TQL\_20241019T173735.SAFE | 0.057356969 | 0.067088827 | 0.077990562 | 0.092113629 | 0.110848315 | 0.131193683 | 0.149707973 | 0.166374207 | 0.181345001 | 0.192484081 | 0.198435113 | 0.202944994 | 0.210099772 | 0.220509827 | 0.233666494 | 0.249332398 | 0.267304152 | 0.287344337 | 0.309588581 | 0.335545897 | 0.366623431 | 0.402532965 | 0.442528576 | 0.485915065 | 0.532370329 | 0.583098114 | 0.638607025 | 0.695641875 | 0.751388192 | 0.807999134 | 0.867441475 | 0.925680161 | 0.978205025 | 1.024880767 | 1.065317154 | 1.093936205 | 1.105583906 | 1.100565434 | 1.080593109 | 1.045514345 | 0.996600568 | 0.941566348 | 0.886973023 | 0.829175234 | 0.76464653 | 0.700423002 | 0.643947721 | 0.593575954 | 0.545442224 | 0.49732548 | 0.448666096 | 0.403177291 | 0.364707589 | 0.332375407 | 0.304417491 | 0.280291289 |

Full\_Spectrum\_std\_FLEX\_table.csv

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| site\_code | latitude | longitude | flex\_date | flex\_time | flex\_filename | s2\_filename | Sif Emission Spectrum\_sif\_wavelength\_grid=670 | Sif Emission Spectrum\_sif\_wavelength\_grid=672 | Sif Emission Spectrum\_sif\_wavelength\_grid=674 | Sif Emission Spectrum\_sif\_wavelength\_grid=676 | Sif Emission Spectrum\_sif\_wavelength\_grid=678 | Sif Emission Spectrum\_sif\_wavelength\_grid=680 | Sif Emission Spectrum\_sif\_wavelength\_grid=682 | Sif Emission Spectrum\_sif\_wavelength\_grid=684 | Sif Emission Spectrum\_sif\_wavelength\_grid=686 | Sif Emission Spectrum\_sif\_wavelength\_grid=688 | Sif Emission Spectrum\_sif\_wavelength\_grid=690 | Sif Emission Spectrum\_sif\_wavelength\_grid=692 | Sif Emission Spectrum\_sif\_wavelength\_grid=694 | Sif Emission Spectrum\_sif\_wavelength\_grid=696 | Sif Emission Spectrum\_sif\_wavelength\_grid=698 | Sif Emission Spectrum\_sif\_wavelength\_grid=700 | Sif Emission Spectrum\_sif\_wavelength\_grid=702 | Sif Emission Spectrum\_sif\_wavelength\_grid=704 | Sif Emission Spectrum\_sif\_wavelength\_grid=706 | Sif Emission Spectrum\_sif\_wavelength\_grid=708 | Sif Emission Spectrum\_sif\_wavelength\_grid=710 | Sif Emission Spectrum\_sif\_wavelength\_grid=712 | Sif Emission Spectrum\_sif\_wavelength\_grid=714 | Sif Emission Spectrum\_sif\_wavelength\_grid=716 | Sif Emission Spectrum\_sif\_wavelength\_grid=718 | Sif Emission Spectrum\_sif\_wavelength\_grid=720 | Sif Emission Spectrum\_sif\_wavelength\_grid=722 | Sif Emission Spectrum\_sif\_wavelength\_grid=724 | Sif Emission Spectrum\_sif\_wavelength\_grid=726 | Sif Emission Spectrum\_sif\_wavelength\_grid=728 | Sif Emission Spectrum\_sif\_wavelength\_grid=730 | Sif Emission Spectrum\_sif\_wavelength\_grid=732 | Sif Emission Spectrum\_sif\_wavelength\_grid=734 | Sif Emission Spectrum\_sif\_wavelength\_grid=736 | Sif Emission Spectrum\_sif\_wavelength\_grid=738 | Sif Emission Spectrum\_sif\_wavelength\_grid=740 | Sif Emission Spectrum\_sif\_wavelength\_grid=742 | Sif Emission Spectrum\_sif\_wavelength\_grid=744 | Sif Emission Spectrum\_sif\_wavelength\_grid=746 | Sif Emission Spectrum\_sif\_wavelength\_grid=748 | Sif Emission Spectrum\_sif\_wavelength\_grid=750 | Sif Emission Spectrum\_sif\_wavelength\_grid=752 | Sif Emission Spectrum\_sif\_wavelength\_grid=754 | Sif Emission Spectrum\_sif\_wavelength\_grid=756 | Sif Emission Spectrum\_sif\_wavelength\_grid=758 | Sif Emission Spectrum\_sif\_wavelength\_grid=760 | Sif Emission Spectrum\_sif\_wavelength\_grid=762 | Sif Emission Spectrum\_sif\_wavelength\_grid=764 | Sif Emission Spectrum\_sif\_wavelength\_grid=766 | Sif Emission Spectrum\_sif\_wavelength\_grid=768 | Sif Emission Spectrum\_sif\_wavelength\_grid=770 | Sif Emission Spectrum\_sif\_wavelength\_grid=772 | Sif Emission Spectrum\_sif\_wavelength\_grid=774 | Sif Emission Spectrum\_sif\_wavelength\_grid=776 | Sif Emission Spectrum\_sif\_wavelength\_grid=778 | Sif Emission Spectrum\_sif\_wavelength\_grid=780 |
| IT-JDS2 | 44.85315 | 12.06556 | 20220717 | 93225 | PRS\_TD\_20220717\_093225.nc | S2A\_MSIL2A\_20220717T100611\_N0510\_R022\_T32TQQ\_20240717T130024.SAFE | 0.029822 | 0.034805 | 0.040402 | 0.047716 | 0.057493 | 0.068346 | 0.078665 | 0.088538 | 0.097944 | 0.105415 | 0.109934 | 0.11325 | 0.117431 | 0.122777 | 0.129109 | 0.136275 | 0.144289 | 0.153075 | 0.16278 | 0.174104 | 0.187817 | 0.203635 | 0.22111 | 0.239928 | 0.259839 | 0.281443 | 0.304992 | 0.328959 | 0.352036 | 0.375362 | 0.399873 | 0.423773 | 0.445108 | 0.463861 | 0.479955 | 0.490962 | 0.49457 | 0.491032 | 0.48106 | 0.464641 | 0.442279 | 0.417322 | 0.392655 | 0.366615 | 0.337703 | 0.308977 | 0.283784 | 0.261317 | 0.239914 | 0.218589 | 0.19704 | 0.176964 | 0.159989 | 0.145743 | 0.133468 | 0.122864 |
| IT-JDS | 44.87431 | 11.9792 | 20220717 | 93225 | PRS\_TD\_20220717\_093225.nc | S2A\_MSIL2A\_20220717T100611\_N0510\_R022\_T32TQQ\_20240717T130024.SAFE | 0.021673 | 0.02538 | 0.029521 | 0.034891 | 0.042038 | 0.049845 | 0.057046 | 0.063638 | 0.069624 | 0.074024 | 0.076244 | 0.077607 | 0.079757 | 0.082854 | 0.086816 | 0.091579 | 0.097191 | 0.103568 | 0.110788 | 0.11938 | 0.129849 | 0.142153 | 0.155978 | 0.171125 | 0.18755 | 0.20565 | 0.225745 | 0.246616 | 0.26733 | 0.288682 | 0.311326 | 0.333772 | 0.354414 | 0.373097 | 0.389563 | 0.401709 | 0.407552 | 0.407086 | 0.400901 | 0.388855 | 0.371462 | 0.351631 | 0.331903 | 0.310909 | 0.287251 | 0.263646 | 0.242832 | 0.224206 | 0.206421 | 0.188486 | 0.170256 | 0.153181 | 0.138699 | 0.126471 | 0.115889 | 0.106721 |
| IT-SR2 | 43.732 | 10.291 | 20230616 | 101431 | PRS\_TD\_20230616\_101431.nc | S2B\_MSIL2A\_20230617T100559\_N0509\_R022\_T32TPP\_20230617T131301.SAFE | 0.043931 | 0.052744 | 0.062625 | 0.075305 | 0.091866 | 0.109247 | 0.123889 | 0.134558 | 0.140329 | 0.139718 | 0.131962 | 0.120687 | 0.109678 | 0.099799 | 0.0911 | 0.083678 | 0.077674 | 0.072956 | 0.06945 | 0.067181 | 0.066144 | 0.065996 | 0.066435 | 0.067288 | 0.068531 | 0.070258 | 0.072476 | 0.074911 | 0.077217 | 0.079607 | 0.082389 | 0.085073 | 0.087406 | 0.089314 | 0.090833 | 0.091558 | 0.091081 | 0.089478 | 0.086873 | 0.083297 | 0.078847 | 0.073963 | 0.069237 | 0.064308 | 0.058878 | 0.053626 | 0.049018 | 0.044926 | 0.041058 | 0.03727 | 0.033465 | 0.029968 | 0.027009 | 0.024576 | 0.022484 | 0.020676 |
| US-LIN | 41.1797 | -96.4404 | 20230711 | 171357 | PRS\_TD\_20230711\_171357.nc | S2A\_MSIL2A\_20230711T171901\_N0510\_R012\_T14TQL\_20241019T173735.SAFE | 0.013224 | 0.015435 | 0.017929 | 0.021197 | 0.025504 | 0.030201 | 0.03443 | 0.038293 | 0.041766 | 0.044451 | 0.045966 | 0.047192 | 0.049108 | 0.051811 | 0.055219 | 0.059313 | 0.064028 | 0.06927 | 0.075137 | 0.081988 | 0.090173 | 0.099652 | 0.110279 | 0.121891 | 0.13441 | 0.148216 | 0.163378 | 0.179112 | 0.194638 | 0.210577 | 0.227346 | 0.243883 | 0.258982 | 0.27255 | 0.284401 | 0.293085 | 0.297109 | 0.296561 | 0.291835 | 0.282883 | 0.270062 | 0.255479 | 0.240986 | 0.225594 | 0.208321 | 0.191055 | 0.175856 | 0.162277 | 0.149277 | 0.136233 | 0.122982 | 0.110627 | 0.100118 | 0.091301 | 0.083607 | 0.077009 |

L2B\_FLEX\_FLOX\_matchup.csv

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| site\_code | date | SIF\_FARRED\_max\_flox | SIF\_FARRED\_max\_wvl\_flox | SIF\_RED\_max\_flox | SIF\_RED\_max\_wvl\_flox | SIF\_O2B\_flox | SIF\_O2A\_flox | SIF\_int\_flox | SIF\_FARRED\_max\_un\_flox | SIF\_FARRED\_max\_wvl\_un\_flox | SIF\_RED\_max\_un\_flox | SIF\_RED\_max\_wvl\_un\_flox | SIF\_O2B\_un\_flox | SIF\_O2A\_un\_flox | SIF\_int\_un\_flox | SIF\_FARRED\_max\_flex | SIF\_FARRED\_max\_wvl\_flex | SIF\_RED\_max\_flex | SIF\_RED\_max\_wvl\_flex | SIF\_O2B\_flex | SIF\_O2A\_flex | SIF\_int\_flex | SIF\_FARRED\_max\_un\_flex | SIF\_FARRED\_max\_wvl\_un\_flex | SIF\_RED\_max\_un\_flex | SIF\_RED\_max\_wvl\_un\_flex | SIF\_O2B\_un\_flex | SIF\_O2A\_un\_flex | SIF\_int\_un\_flex | SIF\_O2A\_tf | SIF\_FARRED\_max\_tf | SIF\_int\_tf | SIF\_O2B\_tf | SIF\_RED\_max\_tf | SIF\_O2A\_un\_tf | SIF\_FARRED\_max\_un\_tf | SIF\_int\_un\_tf | SIF\_O2B\_un\_tf | SIF\_RED\_max\_un\_tf |
| IT-JDS | 20220717 | 0.816375 | 740.123 | 0.154922 | 685.241 | 0.167197 | 0.514399 | 44.70749 | 0.083364 | 0.152 | 0.017218 | 0.161 | 0.018446 | 0.053166 | 4.660647 | 0.816375 | 740.123 | 0.154921 | 685.241 | 0.167196 | 0.514399 | 44.70749 | 0.083364 | 0.152 | 0.017218 | 0.161 | 0.018446 | 0.053166 | 4.660646 | 2.170055 | 3.443978 | 188.6041 | 2.276627 | 2.109485 | 0 | 0 | 0 | 0 | 0 |
| US-LIN | 20230711 | 1.093936 | 740.123 | 0.17386 | 685.241 | 0.186915 | 0.700423 | 58.20037 | 0.11112 | 0.152 | 0.019112 | 0.161 | 0.020418 | 0.071769 | 6.009934 | 1.093936 | 740.123 | 0.17386 | 685.241 | 0.186915 | 0.700423 | 58.20037 | 0.11112 | 0.152 | 0.019112 | 0.161 | 0.020418 | 0.071769 | 6.009934 | 0.548502 | 0.856662 | 45.57673 | 0.152915 | 0.142234 | 0 | 0 | 0 | 0 | 0 |
| IT-SR2 | 20230616 | 1.587888 | 740.123 | 0.789807 | 685.241 | 0.837398 | 0.933361 | 104.6114 | 0.160515 | 0.152 | 0.080707 | 0.161 | 0.085466 | 0.095062 | 10.65103 | 1.587888 | 740.123 | 0.789807 | 685.241 | 0.837398 | 0.93336 | 104.6114 | 0.160515 | 0.152 | 0.080707 | 0.161 | 0.085466 | 0.095062 | 10.65103 | 0.97342 | 1.65604 | 109.1013 | 1.021999 | 0.963916 | 0 | 0 | 0 | 0 | 0 |
| IT-JDS2 | 20220717 | 1.093936 | 740.123 | 0.17386 | 685.241 | 0.186915 | 0.700423 | 58.20037 | 0.11112 | 0.152 | 0.019112 | 0.161 | 0.020418 | 0.071769 | 6.009934 | 0.842756 | 657.8871 | 0.146618 | 609.1032 | 0.158643 | 0.53677 | 45.57831 | 0.086002 | 0.135303 | 0.016388 | 0.143303 | 0.017591 | 0.055403 | 4.747728 | N/A | N/A | N/A | N/A | N/A | 0 | 0 | 0 | 0 | 0 |

L2B\_FLEX\_table.csv

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| site\_code | latitude | longitude | flex\_date | flex\_time | flex\_filename | s2\_filename | SIF\_FARRED\_max | SIF\_FARRED\_max\_wvl | SIF\_RED\_max | SIF\_RED\_max\_wvl | SIF\_O2B | SIF\_O2A | SIF\_int | SIF\_FARRED\_max\_un | SIF\_FARRED\_max\_wvl\_un | SIF\_RED\_max\_un | SIF\_RED\_max\_wvl\_un | SIF\_O2B\_un | SIF\_O2A\_un | SIF\_int\_un |
| IT-JDS2 | 44.85315 | 12.06556 | 20220717 | 93225 | PRS\_TD\_20220717\_093225.nc | S2A\_MSIL2A\_20220717T100611\_N0510\_R022\_T32TQQ\_20240717T130024.SAFE | 0.842756 | 657.8871 | 0.146618 | 609.1032 | 0.158643 | 0.53677 | 45.57831 | 0.086002 | 0.135303 | 0.016388 | 0.143303 | 0.017591 | 0.055403 | 4.747728 |
| IT-JDS | 44.87431 | 11.9792 | 20220717 | 93225 | PRS\_TD\_20220717\_093225.nc | S2A\_MSIL2A\_20220717T100611\_N0510\_R022\_T32TQQ\_20240717T130024.SAFE | 0.816375 | 740.123 | 0.154921 | 685.241 | 0.167196 | 0.514399 | 44.70749 | 0.083364 | 0.152 | 0.017218 | 0.161 | 0.018446 | 0.053166 | 4.660646 |
| IT-SR2 | 43.732 | 10.291 | 20230616 | 101431 | PRS\_TD\_20230616\_101431.nc | S2B\_MSIL2A\_20230617T100559\_N0509\_R022\_T32TPP\_20230617T131301.SAFE | 1.587888 | 740.123 | 0.789807 | 685.241 | 0.837398 | 0.93336 | 104.6114 | 0.160515 | 0.152 | 0.080707 | 0.161 | 0.085466 | 0.095062 | 10.65103 |
| US-LIN | 41.1797 | -96.4404 | 20230711 | 171357 | PRS\_TD\_20230711\_171357.nc | S2A\_MSIL2A\_20230711T171901\_N0510\_R012\_T14TQL\_20241019T173735.SAFE | 1.093936 | 740.123 | 0.17386 | 685.241 | 0.186915 | 0.700423 | 58.20037 | 0.11112 | 0.152 | 0.019112 | 0.161 | 0.020418 | 0.071769 | 6.009934 |

L2B\_log\_report.csv

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| site\_code | latitude | longitude | reference\_area | time\_window | threshold\_CV | vegetation\_pixel | threshold\_cloud | flex\_date | flex\_time | flex\_filename | flex\_valid\_pixels | s2\_filename | s2\_date | s2\_time | time\_difference\_s2\_flex | s2\_valid\_pixels | s2\_ndvi\_avg | s2\_ndvi\_sd | s2\_ndvi\_cv | s2\_ndvi\_cv\_flag | s2\_nirv\_avg | s2\_nirv\_sd | s2\_nirv\_cv | s2\_nirv\_cv\_flag | note |
| IT-CP2 | 41.7043 | 12.3573 | 900 | 15 | 20 | 50 | 50 | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | No FLOX data on the same date 20230824 |
| IT-JDS | 44.87431 | 11.9792 | 900 | 15 | 20 | 50 | 50 | 20220717 | 93225 | PRS\_TD\_20220717\_093225.nc | 100 | S2A\_MSIL2A\_20220717T100611\_N0510\_R022\_T32TQQ\_20240717T130024.SAFE | 20220717 | 100611 | 0 | 100 | 0.390276 | 0.299385 | 0.767112 | 0 | 0.150361 | 0.148476 | 0.987462 | 0 | N/A |
| US-LIN | 41.1797 | -96.4404 | 900 | 15 | 20 | 50 | 50 | 20230711 | 171357 | PRS\_TD\_20230711\_171357.nc | 100 | S2A\_MSIL2A\_20230711T171901\_N0510\_R012\_T14TQL\_20241019T173735.SAFE | 20230711 | 171901 | 0 | 100 | 0.735312 | 0.26231 | 0.356733 | 0 | 0.284885 | 0.121498 | 0.42648 | 0 | N/A |
| IT-SR2 | 43.732 | 10.291 | 900 | 20 | 20 | 50 | 50 | 20230616 | 101431 | PRS\_TD\_20230616\_101431.nc | 100 | S2B\_MSIL2A\_20230617T100559\_N0509\_R022\_T32TPP\_20230617T131301.SAFE | 20230617 | 100559 | 1 | 100 | 0.621877 | 0.194361 | 0.312539 | 0 | 0.148148 | 0.052576 | 0.354889 | 0 | N/A |
| IT-JDS2 | 44.85315 | 12.06556 | 900 | 20 | 20 | 50 | 50 | 20220717 | 93225 | PRS\_TD\_20220717\_093225.nc | 100 | S2A\_MSIL2A\_20220717T100611\_N0510\_R022\_T32TQQ\_20240717T130024.SAFE | 20220717 | 100611 | 0 | 53.05273 | 0.232402 | 0.238378 | 1.025715 | 0 | 0.100301 | 0.104745 | 1.044311 | 0 | N/A |

L2B\_validation\_report\_flex\_flox.csv

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SIF metrics | n\_sites | n\_images | R^2 | RMSE | Bias | Slope | Intercept | Random uncertainty |
| SIF\_FARRED\_max | 4 | 3 | 0.878271 | 0.12559 | 0.062795 | 0.841313 | 0.235008 | 0.01183 |
| SIF\_FARRED\_max\_wvl | 4 | 3 | 1 | 41.11796 | 20.55899 | 0 | 740.123 | 1268.014 |
| SIF\_RED\_max | 4 | 3 | 0.998331 | 0.013621 | 0.00681 | 0.984558 | 0.011695 | 0.000139 |
| SIF\_RED\_max\_wvl | 4 | 3 | 1 | 38.06889 | 19.03443 | 0 | 685.241 | 1086.931 |
| SIF\_O2B | 4 | 3 | 0.998387 | 0.014136 | 0.007068 | 0.984838 | 0.012186 | 0.00015 |
| SIF\_O2A | 4 | 3 | 0.82141 | 0.081826 | 0.040913 | 0.803938 | 0.172518 | 0.005022 |
| SIF\_int | 4 | 3 | 0.952239 | 6.311031 | 3.155525 | 0.906637 | 9.063016 | 29.87177 |
| SIF\_FARRED\_max\_un | 4 | 3 | 0.878271 | 0.012559 | 0.00628 | 0.841312 | 0.023775 | 0.000118 |
| SIF\_FARRED\_max\_wvl\_un | 4 | 3 | 1 | 0.008349 | 0.004174 | 0 | 0.152 | 5.23E-05 |
| SIF\_RED\_max\_un | 4 | 3 | 0.998331 | 0.001362 | 0.000681 | 0.984557 | 0.001196 | 1.39E-06 |
| SIF\_RED\_max\_wvl\_un | 4 | 3 | 1 | 0.008849 | 0.004424 | 0 | 0.161 | 5.87E-05 |
| SIF\_O2B\_un | 4 | 3 | 0.998387 | 0.001414 | 0.000707 | 0.984838 | 0.001245 | 1.50E-06 |
| SIF\_O2A\_un | 4 | 3 | 0.82141 | 0.008183 | 0.004091 | 0.803937 | 0.01759 | 5.02E-05 |
| SIF\_int\_un | 4 | 3 | 0.952239 | 0.631103 | 0.315551 | 0.906636 | 0.924038 | 0.298719 |

L2B\_validation\_report\_flex\_tf.csv

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SIF metrics | n\_sites | n\_images | R^2 | RMSE | Bias | Slope | Intercept | Random uncertainty |
| SIF\_FARRED\_max | 3 | 3 | 0.300492 | 1.523728 | 0.819494 | -1.85832 | 4.152481 | 1.650178 |
| SIF\_RED\_max | 3 | 3 | 0.014567 | 1.133083 | 0.699016 | -0.33015 | 1.194978 | 0.795254 |
| SIF\_O2B | 3 | 3 | 0.016879 | 1.222692 | 0.753344 | -0.36371 | 1.294968 | 0.92745 |
| SIF\_O2A | 3 | 3 | 0.441923 | 0.960188 | 0.514598 | -2.66274 | 3.137345 | 0.65715 |
| SIF\_int | 3 | 3 | 0.076792 | 83.43812 | 45.25432 | -0.63197 | 158.143 | 4913.967 |
| SIF\_FARRED\_max\_un | 3 | 3 | 1 | 0.122559 | -0.11833 | 0 | 0 | 0.001018 |
| SIF\_RED\_max\_un | 3 | 3 | 1 | 0.048906 | -0.03901 | 0 | 0 | 0.00087 |
| SIF\_O2B\_un | 3 | 3 | 1 | 0.051838 | -0.04144 | 0 | 0 | 0.00097 |
| SIF\_O2A\_un | 3 | 3 | 1 | 0.075309 | -0.07333 | 0 | 0 | 0.000294 |
| SIF\_int\_un | 3 | 3 | 1 | 7.556133 | -7.1072 | 0 | 0 | 6.58279 |