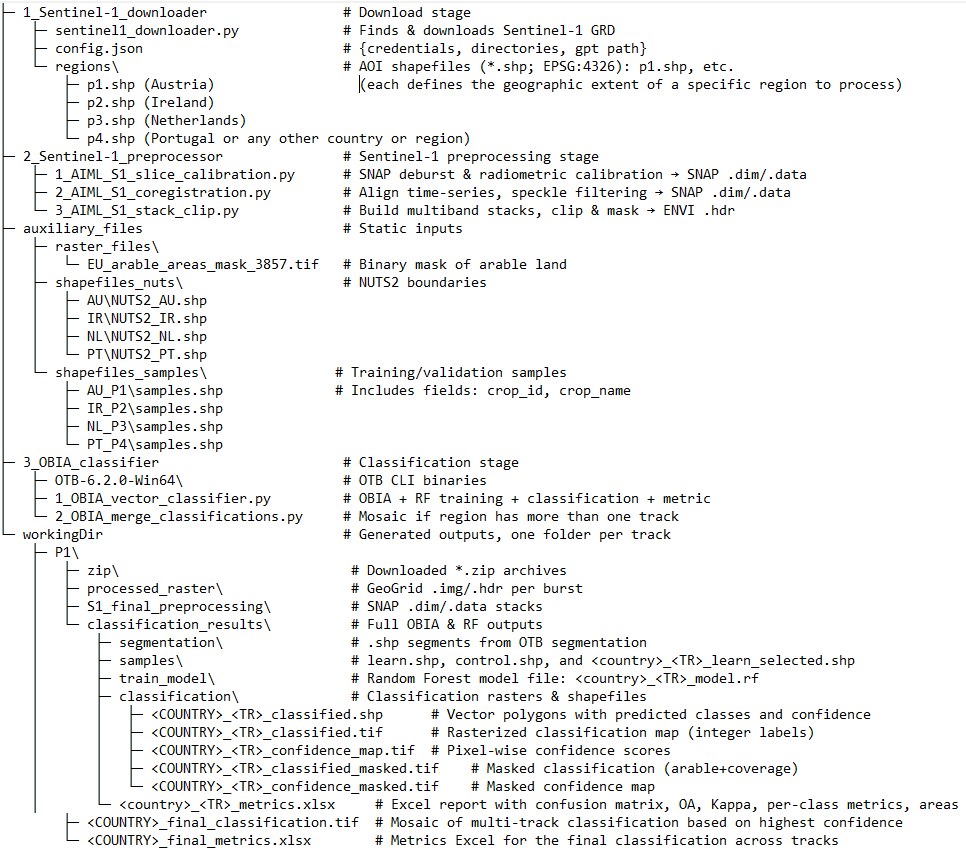
**AIML\_CropMapper: Installation & Usage**

This guide will walk you through setting up an Anaconda environment for AIML\_CropMapper, configuring external tools (ESA SNAP & OrfeoToolbox), and running each stage of the processing pipeline.

**1. Prerequisites**

1. **Windows 10/11** with administrative privileges.
2. **Anaconda/Miniconda** (Python 3.10+).
3. **ESA SNAP** installed via Windows installer (e.g., default <drive>:\Program Files\esa-snap).
4. **OrfeoToolbox (OTB)** at <drive>:\AIML\_CropMapper\3\_OBIA\_classifier\OTB-6.2.0-Win64.
5. Internet access for Sentinel-1 downloads.

**Top-level folder:** <drive>:\AIML\_CropMapper



**2. Conda Environment & Dependencies**

* **Create & activate from Anconda command prompt:**

conda create -n aiml\_cropmapper python=3.10 –y  
conda activate aiml\_cropmapper

* **Install packages:**

conda install -c conda-forge numpy pandas geopandas rasterio gdal fiona shapely pyproj scikit-learn openpyxl requests loguru –y

**3. Setup external tools on on Windows PATH**

* ***Add SNAP GPT path:***

setx PATH "<drive>:\Program Files\esa-snap\bin;%PATH%"

* **Add OTB CLI (Orfeo Toolbox Command-Line Interface) path:**

setx PATH "<drive>:\AIML\_CropMapper\3\_OBIA\_classifier\OTB-6.2.0-Win64\bin;%PATH%"

**4. Setup downloader configuration**

In 1\_Sentinel-1\_downloader/config.json:

{

"creodias\_user\_mail": "your CDSE email",

"creodias\_user\_password": "your CDSE password",

"download\_directory": "<drive>:\\AIML\_CropMapper\\workingDir",

"geometry\_directory": "regions",

"gpt\_directory": "<drive>:\\Program Files\\esa-snap\\bin\\gpt.exe"

}

– Replace with your own credentials to the **Copernicus Data Space Ecosystem**

**5. Stage 1 – Download Sentinel-1 GRD scenes for given track by command:**

conda activate aiml\_cropmapper

cd <drive>:

cd: <drive>:\AIML\_CropMapper\1\_Sentinel-1\_downloader

python sentinel1\_downloader.py -j config.json -s 2024-10-15 -e 2025-10-17 --track P4

**-j**: config file  
**-s/-e**: start/end dates  
**--track**: P1–P4

Outputs .zip in workingDir\P4\zip.

**6. Stage 2 – Sentinel-1 Preprocessing**

This stage processes raw Sentinel-1 .zip products into stacked, clipped, and masked raster data for classification. It comprises three sequential Python scripts. Run these in order.

**6.1 Step 1 – Calibration and Slice Assembly**

**Script**: 1\_AIML\_S1\_slice\_calibration.py  
**Location**: <drive>:\AIML\_CropMapper\2\_Sentinel-1\_preprocessor

**Purpose:**

* Calibrate .zip S1 scenes (thermal noise removal, orbit correction, border noise removal, sigma nought calibration)
* Assemble slices (bursts) into a single .dim product per date
* Clip to polygon using predefined geoRegion (hardcoded per track)

**Input folders:**

* workingDir\<track>\zip\ — contains .zip Sentinel-1 scenes

**Output folders:**

* workingDir\<track>\calibrated\ — temporary .dim files (removed after slicing)
* workingDir\<track>\slice\_assembly\YYYYMMDD\_<track>\_IW\_GRDH\_<Sensor>.dim/.data — final calibrated S1 products for a date

**Run the command:**cd F:\AIML\_CropMapper\2\_Sentinel-1\_preprocessor  
python 1\_AIML\_S1\_slice\_calibration.py -track P4

You can also run for multiple tracks:  
python 1\_AIML\_S1\_slice\_calibration.py -track P1 P2 P3 P4

**What it does internally:**

1. For each .zip scene in zip/, it generates a SNAP XML file dynamically with proper input/output paths.
2. Executes gpt with:  
   gpt -DAuxDataPath=<your SNAP aux path> -q 4 <calibration\_graph.xml>
3. After all .zip scenes are calibrated, it groups .dim files by acquisition date and performs **slice assembly**, generating one .dim per day.
4. Temporary .xml files and intermediate .dim are deleted to save space.

**6.2 Step 2 – Coregistration and Filtering  
Script**: 2\_AIML\_S1\_coregistration.py  
**Location**: <drive>:\AIML\_CropMapper\2\_Sentinel-1\_preprocessor

**Purpose:**

* Stack and coregister all .dim products by acquisition date
* Apply multi-temporal speckle filtering (Lee-Sigma)
* Perform terrain correction and final speckle filtering (Median)
* Generate one VH and one VV .dim file with all stacked bands

**Input folders:**

* workingDir\<track>\slice\_assembly\\*.dim — from previous step

**Output folder:**

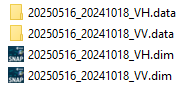
* workingDir\<track>\S1\_final\_preprocessing\YYYYMMDD\_YYYYMMDD\_VH.dim/.data
* workingDir\<track>\S1\_final\_preprocessing\YYYYMMDD\_YYYYMMDD\_VV.dim/.data

**Run the command:**python 2\_AIML\_S1\_coregistration.py --track P4

You can also run for multiple tracks:  
python 2\_AIML\_S1\_coregistration.py --track P1 P3

**What it does internally:**

1. Builds a virtual product list using CreateStack in SNAP for all .dim files (sorted by date)
2. Applies:
   * **Cross-correlation** alignment
   * **Warping** to correct geometry
   * **Multi-temporal speckle filtering**
   * **Terrain correction** using SRTM 3Sec DEM
   * **Log conversion (dB) + Median speckle filter**
3. Outputs two .dim files: one for VH, one for VV, containing multitemporal bands (one per date).



**6.3 Step 3 – Stack Bands, Clip to AOI, and Mask**

**Script**: 3\_AIML\_S1\_stack\_clip.py  
**Location**: <drive>:\AIML\_CropMapper\2\_Sentinel-1\_preprocessor

**Purpose:**

* Read VH and VV .img files from .data folders
* Stack bands into a single multiband .img and .hdr
* Clip raster using NUTS2 shapefile (NUTS2\_<COUNTRY>.shp)
* Output clipped raster per region

**Input folders:**

* workingDir\<track>\S1\_final\_preprocessing\\*VH.data\ and \*VV.data\
* auxiliary\_files\shapefiles\_nuts\<COUNTRY>\NUTS2\_<COUNTRY>.shp

**Output folder:**

* workingDir\<track>\processed\_raster\COUNTRY\_<track>\_YYYYMMDD\_YYYYMMDD\_VH\_VV.img/.hdr

**Run the command:**python 3\_AIML\_S1\_stack\_clip.py --track P4

You can also run for multiple tracks:  
python 3\_AIML\_S1\_stack\_clip.py --track P1 P2 P3 P4

**What it does internally:**

1. Collects all .img slices from VH and VV .data folders
2. Sorts bands by date (from filename) and stacks them in order
3. Clips stacked image to regional shapefile:  
   auxiliary\_files\shapefiles\_nuts\PT\NUTS2\_PT.shp ← for P4/PT
4. Output is stored as .img + .hdr with the name format:  
   PT\_P4\_20241015\_20250508\_VH\_VV.img   
   PT\_P4\_20241015\_20250508\_VH\_VV.hdr
5. Intermediate stack is deleted after clipping to save space.

**7. Stage 3 – OBIA Classification**

**Overall goal:** segment the SAR stack into objects, train/apply a Random Forest on samples, rasterize and mask the results, then compute metrics and export to excel.

**7.1 Run the OBIA pipeline script**cd <drive>:\AIML\_CropMapper\3\_OBIA\_classifierpython 1\_OBIA\_vector\_classifier.py --track P4

* Use --track to list one or more belts (e.g., P1 P2 P3 P4).

**7.2 Stage Breakdown**

1. **Segmentation**
   * **input:** workingDir/<track>/processed\_raster/\*\_VH\_VV\_stack.img
   * **tool:** otbcli\_LargeScaleMeanShift
   * **output:** classification\_results/segmentation/<track>\_segmentation.shp
2. **Split samples (70/30)**
   * **input:** auxiliary\_files/shapefiles\_samples/PT\_P4/samples.shp
   * **output:**
     + classification\_results/samples/learn.shp
     + classification\_results/samples/control.shp
3. **Select training polygons**
   * **input:** learn.shp + segmentation.shp
   * **output:** classification\_results/samples/PT\_P4\_learn\_selected.shp
4. **Train random forest**
   * **input:** \*\_learn\_selected.shp, features meanB\*
   * **tool:** otbcli\_TrainVectorClassifier
   * **output:** classification\_results/train\_model/PT\_P4\_model.rf
5. **Vector classification + confidence**
   * **input:** segmentation.shp, PT\_P4\_model.rf
   * **tool:** otbcli\_VectorClassifier
   * **outputs:**
     + classification/PT\_P4\_classified.shp (with predicted,confidence fields)
6. **Rasterize classification**
   * **input:** PT\_P4\_classified.shp
   * **tool:** otbcli\_Rasterization
   * **output:** classification/PT\_P4\_classified.tif
7. **Rasterize Confidence**
   * **input:** PT\_P4\_classified.shp
   * **output:** classification/PT\_P4\_confidence\_map.tif
8. **Cutline extraction**
   * **input:** the classified raster
   * **method:** GDAL Polygonize in Python
   * **output:** processed\_raster/PT\_P4\_valid\_coverage.shp
9. **Mask Classification**
   * **inputs:**
     + classification/PT\_P4\_classified.tif
     + arable mask: auxiliary\_files/raster\_files/EU\_arable\_areas\_mask\_3857.tif
     + cutline shapefile: processed\_raster/PT\_P4\_valid\_coverage.shp
   * **Output:** classification/PT\_P4\_classified\_masked.tif
10. **Mask Confidence**
    * **Similar inputs** as above
    * **Output:** classification/PT\_P4\_confidence\_masked.tif
11. **Metrics & Excel Export**
    * **Inputs:**
      + classification\_results/samples/control.shp
      + classification/PT\_P4\_classified\_masked.tif
    * **Metrics computed:**
      + confusion matrix
      + overall accuracy (OA)
      + kappa index
      + producer’s accuracy (recall) & user’s accuracy (precision) per class
      + F1-score per class
      + Class area in hectares (pixel count × pixel area)
    * **Output:** classification\_results/PT\_P4\_metrics.xlsx

**8. Stage 4 – Merge multi-track classifications**

**Script:** 2\_OBIA\_merge\_classifications.py  
**Directory:** <drive>:\AIML\_CropMapper\3\_OBIA\_classifier

When your AOI spans more than one Sentinel-1 belt (e.g. P1 + P1a), this script builds a single, highest-confidence mosaic and computes final metrics.

**8.1 Command**cd <drive>:\AIML\_CropMapper\3\_OBIA\_classifierpython 2\_OBIA\_merge\_classifications.py --track P1

--track P1 will automatically discover and merge **P1** and **P1a** outputs.

**8.2 Inputs**

1. **Masked classification rasters**

* workingDir\<TR>\classification\_results\classification\<COUNTRY>\_<TR>\_classified\_masked.tif
* workingDir\<TR>\classification\_results\classification\<COUNTRY>\_<TR>\_confidence\_masked.tif

1. **Control samples shapefile**
   * workingDir\<TR>\classification\_results\samples\control.shp

**8.3 Workflow**

1. **Discover tracks**
   * Scans workingDir for subfolders beginning with the prefix (P1, P1a)
   * Matches them to country codes (TRACK\_REGIONS)
2. **Compute Global Grid**
   * Opens the first track’s masked classification to read its projection and geotransform
   * Computes the union of all track extents → determines global cols, rows, and gt\_global
3. **Warp & Stack**
   * For each discovered track:
     + applies gdal.Warp to resample both classification and confidence into the global grid, in memory
     + reads the warped arrays into two stacks: class\_stack (int labels) and conf\_stack (float confidences)
4. **Select Highest-Confidence Pixel**
   * Converts any NaN in conf\_stack to -inf so they’re never chosen
   * Computes idx = argmax(conf\_stack, axis=0) to find which track has the highest confidence at each pixel
   * Builds final = take\_along\_axis(class\_stack, idx) as the merged classification
   * Sets any pixel where **all** confidences were nodata back to 0
5. **Save Final Mosaic**
   * Writes classification\_results\<COUNTRY>\_final\_classification.tif with gt\_global and the shared projection
   * Sets the no-data value to 0
6. **Compute Final Metrics**
   * Loads samples\control.shp for the *first* track folder (they’re identical across P1/P1a)
   * For each control point, uses the inverse geotransform to sample final → pairs of true vs. predicted labels
   * Calculates:
     + **Confusion matrix**
     + **Overall Accuracy (OA)**
     + **Kappa**

* Producer’s (recall) and User’s (precision) per class
  + - **F1-score** per class
    - **Class area (ha)** by counting pixels × pixel area

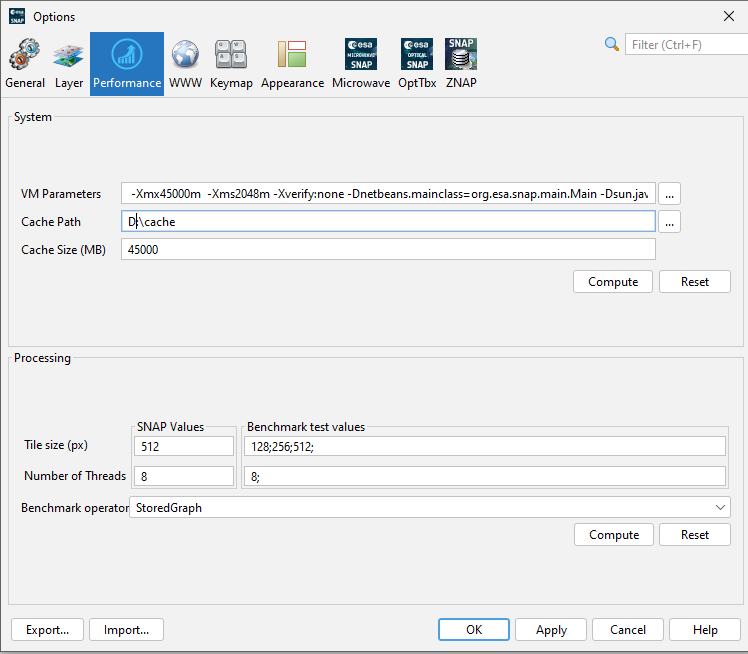
1. **Export Final Excel**
   * Creates classification\_results\<COUNTRY>\_final\_metrics.xlsx with:
     + Table: Confusion matrix
     + OA & Kappa fields
     + Per-class metrics table (Producer’s, User’s, F1)
     + Areas (ha) per class

**8.4 Outputs**

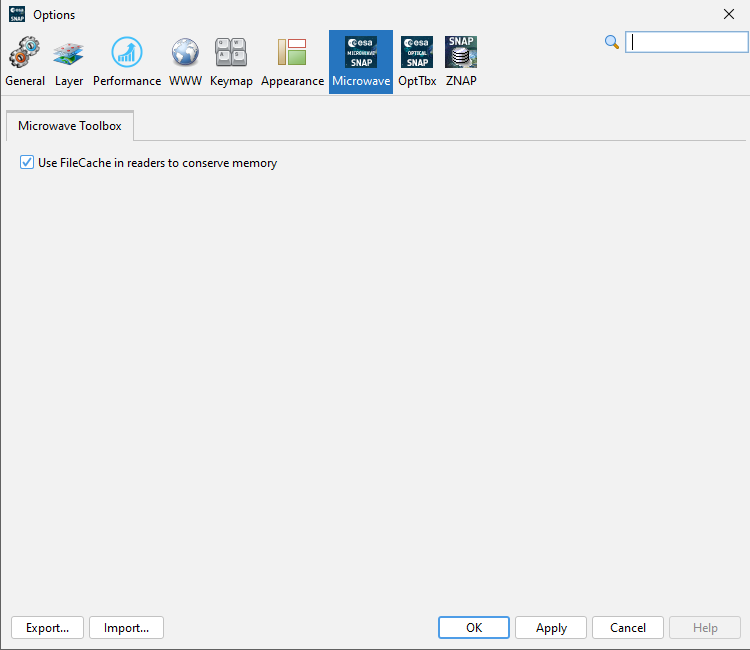
* workingDir\P1\classification\_results\AU\_final\_classification.tif
* workingDir\P1\classification\_results\AU\_final\_metrics.xlsx

**9 Additional performance settings:**

* SNAP (ESA’s Sentinel Application Platform)
  1. **Install location**  
     Don’t install SNAP on your system drive (usually C:) — put it on a dedicated data or applications drive (e.g. D:\ or E:). This avoids I/O contention with your OS and other installed software.
  2. **VM Parameters (**-Xmx **/** -Xms**)**  
     Under **Tools → Options → Performance**, set your Java heap size to about **70 % of your total RAM**. If you have **64 GB RAM**, use -Xmx45000m (≈ 45 GB). You can also set -Xms (initial heap) the same as -Xmx to avoid dynamic resizing pauses.
  3. **Cache Path**  
     Point SNAP’s on-disk cache to a fast drive (SSD) with plenty of free space. E.g. D:\snap-cache.
  4. **Cache Size (MB)**  
     Also in **Options → Performance**, set **Cache Size** to ~70 % of RAM. For 64 GB that’s around **45 000 MB**. A larger cache means fewer disk reads when you revisit tiles or computation intermediates.
  5. **Tile Size (px)**  
     In the **Processing** section of the same dialog, choose a tile size that balances per‐tile overhead vs. memory use. **512 px** is a common sweet spot: small enough to avoid giant in-memory grids, large enough to amortize per-tile overhead.
  6. **Number of Threads**  
     Set this to your **maximum number of CPU cores** (or slightly below, to leave room for OS/jvm threads). If you have a 16-core machine, enter **16**.
  7. **Benchmark Operator**  
     If you run the built-in benchmark (click **Compute** next to “Benchmark operator”), SNAP will suggest optimal tile sizes and thread counts based on your hardware. Re-run it whenever you change major settings.



* 1. Under **Tools → Options → Microwave** check Use FileCache in readers to conserve memory



* OrfeoToolbox (OTB) CLI segmentation

In your **1\_OBIA\_vector\_classifier.py** you call:

run\_cmd(

f"otbcli\_LargeScaleMeanShift "

f"-in {ras} "

f"-spatialr 15 "

f"-ranger 6 "

f"-minsize 50 "

f"-tilesizex 8192 "

f"-tilesizey 8192 "

f"-mode vector "

f"-mode.vector.out {seg\_shp} "

f"-cleanup false "

f"-ram 100128",

1,

'Image segmentation'

)

* 1. **-spatialr (spatial radius)**Defines the radius (in pixels) of the local window used to compute mean-shift clusters in the spatial domain. Larger values gather information from a wider neighborhood, producing smoother, larger segments—but cost a lot of more time to work per pixel (window grows quadratically). Smaller values yield finer detail but can over-segment noisier areas and run faster.
  2. **-tilesizex & -tilesizey**Break the input raster into tiles of size X × Y pixels: here 8192×8192. Larger tiles reduce edge-buffer overhead (fewer overlaps between tiles) but consume more RAM per tile. Smaller tiles lighten per-tile memory use but increase the total number of tiles and thus processing overhead (more blocks to schedule, more repeated reads at tile edges).
  3. **-ram**Specifies how much memory (in MB) OTB may use. Set this to around 70 % of your total RAM (e.g. 45 000 MB on a 64 GB machine) to give plenty of headroom for both OTB and any other concurrent processes.