gee_pipeline — User Cheat Sheet

First time users

Installing local libraries

1. Unzip the folder gee_pipeline somewhere. I recommend having a dedicated folder for your local python modules.

```
C:\Users\1ricardo\my_python_modules\gee_pipeline\
│  pyproject.toml
│  README_DEV.md
└─ gee_pipeline\
   ├─ __init__.py
   ├─ config.py
   ├─ runner.py
   ├─ panels.py
   ├─ ... (others .py)
```

2. Open the terminal PowerShell: Win → “PowerShell” → Enter

IMPORTANT: The file pyproject.toml needs to be exactly in the directory where you'll run the commands pip install

3. Access the project folder

```
cd C:\Users\1ricardo\my_python_modules\gee_pipeline
```

4. Run the dev install python -m (once at a time)

```
python -m pip install -e .
python -m pip install -U pip
python -m pip install -e .[sof]
python -m pip install -e .[silo]
```

If you successfully accomplished this, your environment should now be able to access your local libraries. You don’t need to do this process again.

gee_pipeline — User Cheat Sheet

Optical data

One-minute setup (typical notebook)

1) `from gee_pipeline import Config, run_pipeline_quick`

2) Fill Config with your area, dates, indices you want, and outputs folder.

3) Run: `report = run_pipeline_quick(cfg); print(report.summary_text())`

Minimal example

```
from gee_pipeline import Config, run_pipeline_quick
cfg = Config(
    area_name = "MyFarm",
    yield_year = 2022,
    roi_path = "Contours/MyFarm.gpkg",
    date_start = "2022-06-01",
    date_end = "2022-08-31",
    indices = ["NDVI","EVI"],
    export_root = "Outputs",
    pixel_scale = 10,
    preview = False
)
report = run_pipeline_quick(cfg)
print(report.summary_text())
```

What datasets are used

- Sentinel-2 Surface Reflectance (Harmonized): COPERNICUS/S2_SR_HARMONIZED
- Default cloud/cirrus masking via QA60.
- Canonical band names inside the pipeline:
  BLUE=*B2*, GREEN=*B3*, RED=*B4*, NIR=*B8* (10 m)
  RE1=*B5*, RE2=*B6*, RE3=*B7*, RE4=*B8A*, SWIR1=*B11*, SWIR2=*B12* (20 m)

What you can request in Config.indices (type the names exactly)

10 m-native indices (recommended `pixel_scale = 10`)

NDVI	normalized difference (NIR, RED)
EVI	uses BLUE, RED, NIR
EVI2	two-band EVI (NIR, RED)
GNDVI	NIR vs GREEN
GCI	(NIR/GREEN)-1
NDWI	(GREEN, NIR)
SAVI	soil-adjusted (NIR, RED; L=0.5)
MSAVI2	modified SAVI (NIR, RED)
WDRVI	weighted difference (NIR, RED; a=0.1)

20 m-native indices (recommended `pixel_scale = 20`)

NDRE	NIR vs RE4 (B8A)
CIre	(NIR/RE4)-1

NDMI	NIR vs SWIR1 (B11)
NBR	NIR vs SWIR2 (B12)
MNDWI	GREEN vs SWIR1

Important notes about resolution

- If you include any 20 m indices, set `cfg.pixel_scale = 20` for native resolution.
- If you keep `pixel_scale = 10`, GEE will resample 20 m bands to 10 m internally (interpret accordingly).

Outputs (per run)

- <area>_<year>_cube_stats.parquet (and optional .csv): one-row means per band/date over your ROI.
- <area>_<year>_samples.parquet (and optional .csv): up to sample_size random pixels (lon/lat + bands/indices).
- Logs: Outputs/<area>/<year>/logs/gee_pipeline.log

Common tweaks

- indices: choose any subset from the lists.

```
cfg.indices = ["NDVI","EVI","GNDVI"]
cfg.indices = ["NDVI","NDRE","NDMI"]        # mix; set pixel_scale = 20
```

- sample_size: default cap is 5000. Increase if needed, but be aware of system requirements.

```
cfg.sample_size = 20000
```

- CSV export along with Parquet: cfg.make_csv = True
- CRS of ROI: any valid CRS; the loader reprojects to EPSG:4326 automatically.

Do not do (current package scope)

- Do not change `cfg.collections['s2']` away from COPERNICUS/S2_SR_HARMONIZED.
- Do not request SAR or climate variables here. This package version is optical-only.

gee_pipeline — User Cheat Sheet

SLGA on GEE

One-minute setup (typical notebook)

1) `from gee_pipeline import SLGAPointsConfig, slga_points_quick`
2) Fill Config with your points layer, attributes, depths, and outputs folder.
3) Run: `res = slga_points_quick(cfg); print(res)`

Minimal example

```
from gee_pipeline import SLGAPointsConfig, slga_points_quick
cfg = SLGAPointsConfig (
    area_name = "AUS_Plots",
    points_path = r" Points_SLGA/points.gpkg",
    attributes = ["SOC","CLY"],
    stat = "EV"   # or 05, 95
    depths = [000_005","005_015","015_030"],
    export_root = "Outputs",
    make_parquet = True,
    make_csv = True
)
res = slga_points_quick(cfg)
print(res)
```

What datasets are used

- SLGA on Google Earth Engine: CSIRO/SLGA
- Spatial resolution: ~90 m (3 arc-sec). Native scale: 90.
- Attributes at different depths and predictions percentiles

What you can request in Config.attributes (type the names exactly)

BDW	Bulk density (whole soil)
SOC	Organic carbon
CLY	Clay (<2 µm)
SLT	Silt (2–20 µm)
SND	Sand (20 µm–2 mm)
pHc	pH ($CaCl_2$, 1:5)
AWC	Available Water Capacity
NTO	Total nitrogen.
PTO	Total phosphorus
ECE	Effective Cation Exchange Capacity
DES	Depth of Soil
DER	Depth of Regolith

- Depth intervals: 000_005, 005_015, 015_030, 030_060, 060_100, 100_200 (cm).
- Statistics (stat): EV (estimated value), 05 (5th percentile), 95 (95th percentile).

Check SLGA naming conventions for more info: GSM File Naming Conventions

Important notes about resolution

- The product is static (~90 m). There is no date range to set.
- Keep `scale=90` for native values. If you change scale, values are resampled.
- Points outside Australia or over water return `NaN`.

Outputs (per run)

- Outputs/<area_name>/SLGA/<area>_SLGA_POINTS_<stat>_<timestamp>.parquet (and optional .csv).
- Columns include: your original point fields (e.g., sample_id), lon, lat, and requested SLGA bands.

Common tweaks

- `make_csv`: set to `True` to also export CSV alongside Parquet.
- CRS: the points file must have a defined CRS (the tool reprojects to WGS84 internally).
- Large jobs: for very large point sets (> 100k), consider batching files later or adding a Drive export mode.

Do not do (current package scope)

- Do not pass polygons/lines (use points only; non-point geometries are centroided).
- Do not change the Earth Engine dataset ID (hard-coded to CSIRO/SLGA).
- Do not set date_start/date_end (SLGA is static); they are irrelevant here.

gee_pipeline — User Cheat Sheet

SLGA on COGs - SOF

One-minute setup (typical notebook)

1) `from gee_pipeline import SLGAPointsConfig, sof_points_quick`
2) Fill Config with your points layer, families, fractions, depths, outputs folder and **cookies file**.
3) Run: `res = sof_points_quick(cfg); print(res)`

Minimal example

```
from gee_pipeline import SLGAPointsConfig, slga_points_quick
cfg = SLGAPointsConfig (
    area_name = "SOF_Plots",
    points_path = r" Points_SLGA/points.gpkg",
    families =[Fractions_Density","Proportions","Stocks],
    fractions = ["´MAOC","´POC","´PyOC"]
    depths = [000_005","005_015","015_030"],
    stat = EV
    export_root    =    "Outputs",
    make_parquet = True,
    make_csv = True,
    cookie_file = r"cookies_data.tern.org.au.txt"
)
res = sof_points_quick(cfg
print(res)
```

Authentication process via cookies file

These datasets require authentication to be accessed. You will need to provide your code with cookies file after logging into the system, and then you will be able to parse the data without problems. Follow these steps:

1. Install the Get cookies.txt extension in your browser.
2. Access TERN - Data
3. Click on any. tif file from the collection; you will be redirected to the login page.
4. Authenticate and go back to the data.tern.org.au tab (the same one where you opened .tif).
5. Open the Get cookies.txt extension in the browser bar and export only the cookies from the data.tern.org.au domain (Netscape format).
6. Do not edit the file. You can save it in any folder; I recommend using an absolute path (ex.: C:\Users\1ricardo\tern_cookies.txt).
7. In the code, pass the file path in the `cfg.cookie_file` argument.

Run the script normally - you should be able to access the COGs without 401.

IMPORTANT: cookies expire. If it fails again, generate a new tern_cookies.txt.

Common log erros

Error 401: Unauthorized. Either you did not authenticate, or the cookies have expired. Redo the process of authentication/exporting cookies.

Error 404: Unpublished dataset for the family fraction combination depth stat. The module already skips these combinations and records [skip] ... not published in the log. Check if you have requested something that does

not exist.

Useful extras

The cookie file **must contain data.tern.org.au** domain entries and be in Netscape format (the extension generates this automatically).
Security: do not share your tern_cookies.txt; it authenticates your session.

What datasets are used

- Soil and Landscape Grid National Soil Attribute Maps - Soil Organic Carbon Fractions
- Spatial resolution: ~90 m (3 arc-sec). Native scale: 90.
- Attributes at different depths and predictions percentiles

What you can request in Config.attributes (type the names exactly)

Check SLGA releases in the website: CSIRO Data Access Portal - Soil and Landscape Grid National Soil Attribute Maps - Soil Organic Carbon Fractions (3" resolution) - Release 1

Outputs (per run)

- Outputs/<area_name>/SOF/<area>_SOF_POINTS_<stat>_<timestamp>.parquet (and optional .csv).
- Columns include: your original point fields (e.g., sample_id), lon, lat, and requested bands.

Common tweaks

- `make_csv`: set to `True` to also export CSV alongside Parquet.
- CRS: the points file must have a defined CRS (the tool reprojects to WGS84 internally).
- Large jobs: for very large point sets (> 100k), consider batching files later or adding a Drive export mode.

Do not do (current package scope)

- Do not pass polygons/lines (use points only; non-point geometries are centroided).

gee_pipeline — User Cheat Sheet

SILO API

One-minute setup (typical notebook)

4) `from gee_pipeline import SILOPointsConfig, silo_points_quick`
5) Fill Config with your points layer, variables, dates and id.
6) Run: `report = silo_points_quick(cfg); print(report)`

Minimal example

```
from gee_pipeline.silo import SILOPointsConfig, silo_points_quick
cfg = SILOPointsConfig(
      area_name = "AUS_SILO",
      mode = "datadrill",# 'datadrill' (grid cell) ou 'station'
      points_path = r"Points_SLGA/points.gpkg",
      lat_field = None,
      lon_field = None,  # just use if points_path has no geometry
      station_field = None,   # use for mode='station'
      variables = ["R","X","N","J"],      # SILO code (check website or CheatSheet)
      date_start = "2015-01-01",
      date_end = "2015-12-31",
      username = "seu_email@uni.edu.au",  # mandatory
      password = "apirequest",      # mandatory just w/ DataDrill
      fmt = "csv",      # 'csv' ou 'json'
      export_root = "Outputs",
      make_parquet = True,
      make_csv = True
)
report = silo_points_quick(cfg)
print(report)
```

Authentication process via username and password

Mandatory according to SILO API. However, the first tests worked with a generical username and password.

What datasets are used

- SILO Patch Pointed Dataser available via API < API Tutorial | LongPaddock | Queensland Government>

What you can request in Config.attributes (type the names exactly)

Climate variables using specific names on API requests. Check current variables at: Climate Variables | LongPaddock | Queensland Government

Outputs (per run)

- Outputs/<area_name>/SILO/<area>_SILO_POINTS_<stat>_<timestamp>.parquet (and optional .csv).

Common tweaks

- `make_csv`: set to `True` to also export CSV alongside Parquet.
- CRS: the points file must have a defined CRS (the tool reprojects to WGS84 internally).

- Large jobs: for very large point sets (> 100k), consider batching files later or adding a Drive export mode.
- Variables: pass codes as a list, e.g. ["R","X","N","J"].
- Grid snap (DataDrill): lat/lon are rounded to 0.05°; check lat_snapped/lon_snapped in the output.
- Batching: split large point sets into subsets (by site/region) to avoid timeouts.
- IDs: include a unique point ID in your input; the tool adds source_tag.

Do not do (current package scope)

- Don’t assume sub-grid location accuracy, DataDrill returns the nearest 0.05° cell.
- Don’t pass polygons/lines (points only; others are centroided).
- Don’t forget a valid CRS on your points (reprojected to WGS84 internally).