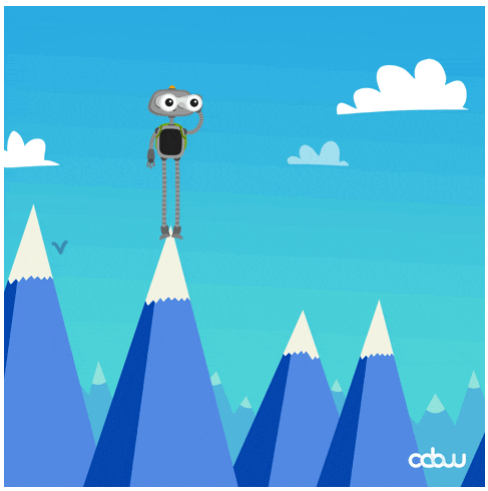
CAIROS — ModelChain (2025-09 update)

handle with care 🚧 **work in progress**



🏔️ Avalanche Scenario Robot incoming ...

NOTE: This README reflects the current workflow, layout, and helpers of the **CAIROS ModelChain**:

- Main script: `cairosModelChain/runCairos.py`

Repo layout

```
Cairos/
├── cairosMapper/        # Mapper tools
└── cairosModelChain/    # Full model chain (main workflow)
```

Quick start (Linux)

1. Minimal system prerequisites

- Install only the flexible, OS-level tools system-wide.
- All project dependencies (AvaFrame, CAIROS, GDAL, NumPy, etc.) will live inside Pixi-managed environments.
- System Python (python3.10 or 3.11 on Ubuntu) is kept minimal — used only for tools like VS Code, Pixi bootstrap, etc.
- CAIROS and AvaFrame never touch system Python → they live in isolated .pixi/envs/* environments.
- Optional:
  - Install VS Code for editing, debugging, and integrated terminals: https://code.visualstudio.com/download

```
# System-wide basics
sudo apt update
sudo apt install -y git python3 python3-pip

# Install Pixi (recommended via installer script)
curl -fsSL https://pixi.sh/install.sh | bash
# restart your shell so `pixi` is in PATH
```

2) Install AvaFrame (temporary: dev branch)

```
# choose your workspace directory, e.g. ~/Documents/Applications
cd ~/Documents/Applications
```

```
# clone AvaFrame
git clone https://github.com/avaframe/AvaFrame.git
cd AvaFrame

# checkout branch until next release (ATM)
git checkout PS_FP_outputRelInfo
```

Two options from here:

Option A: Use AvaFrame via CAIROS

- Nothing else required — CAIROS links AvaFrame in editable mode automatically through Pixi.
- Skip the manual build step.

Option B: Use AvaFrame standalone

- If you want to run AvaFrame directly (outside CAIROS), you need to compile the Cython parts:

```
# set the avalanche dir in local_avaFrameCfg.py and apply your settungs to
local_com4FlowPyCfg.ini
cd AvaFrame
pixi shell
python setup.py build_ext --inplace
pixi shell --environment dev
cd AvaFrame/avaframe
python runCom4FlowPy.py
```

- Repeat this step whenever Cython code changes or after pulling new updates.

3) Install CAIROS repo

```
# choose your workspace directory next to AvaFrame
cd ~/Documents/Applications
git clone <your-cairos-repo> Cairos
cd Cairos
```

Repo layout after clone:

```
Cairos/
├── cairosMapper/
└── cairosModelChain/
```

4) Setup CAIROS ModelChain env

```
cd Cairos

# Clean any old envs if something is corrupted
rm -f pixi.lock
pixi clean -e dev || true
pixi clean cache || true
rm -rf .pixi

# Install dev env (with local AvaFrame)
pixi install -e dev

# Check that CAIROS uses your local AvaFrame
pixi shell -e dev
python -c "import avaframe, pathlib; print(pathlib.Path(avaframe.__file__).resolve())"
```

```
> .../Documents/Applications/AvaFrame/avaframe/__init__.py
```

5) Configure

Copy the defaults and edit the **local** copies:

```
# CAIROS ModelChain config
cd Cairos/cairosModelChain
cp cairosCfg.ini local_cairosCfg.ini

# AvaFrame FlowPy config
cd ../AvaFrame/avaframe/com2FlowPy
cp flowPyAvaFrameCfg.ini local_flowPyAvaFrameCfg.ini
```

6) Run the workflow

```
pixi run -e dev cairos
```

Repository layout (outdated)

```
Cairos/cairosModelChain
├── README.md
├── cairosCodingStandards.md      ← coding guidelines
├── cairosDirectory.md            ← directory description
├── pyproject.toml                ← pixi project manifest
├── pixi.lock                     ← pixi lockfile
├── .pixi/                        ← pixi env folder (hidden)
├── .vscode/                      ← VSCode project settings (hidden)
│   └── settings.json
└── cairos/
    ├── runCairos.py              ← main driver (Steps 00–12)
    ├── runCairosFlowPy.py        ← FlowPy runner (Step 10)
    ├── runInitWorkDir.py         ← initialize work dir (Step 00)
    ├── runPlots.py               ← plotting entrypoint (not used ATM)
    ├── cairosCfg.ini             ← main CAIROS INI (copy to local_*)
    ├── com1PRA/                  ← PRA processing (Steps 01–08)
    │   ├── praDelineation.py
    │   ├── praSelection.py
    │   ├── praSubCatchments.py
    │   ├── praProcessing.py
    │   ├── praSegmentation.py
    │   ├── praAssignElevSize.py
    │   ├── praPrepForFlowPy.py
    │   ├── praMakeBigDataStructure.py
    │   └── plottingUtils.py
    ├── in1Utils/
    │   ├── cfgUtils.py           ← config IO + manifest/effective writer
    │   ├── dataUtils.py          ← raster IO, compression, helpers
    │   └── workflowUtils.py      ← workflow flags, case names, DEM logic
    ├── in2Parameter/
    │   ├── compParams.py         ← FlowPy parameterization + size (Steps 09 & 11)
    │   └── sizeParameters.py
    ├── outPlots/                 ← plotting utils (not used ATM)
    ├── helpBash/
    │   └── printIni              ← bash util for effective INI print (good for debug)
    └── helpPy/
        ├── showDir.py            ← pretty-print project dir
```

```
        └── tifDiff.py           ← calc raster difference utility
├── jupyterLaps/                 ← Jupyter notebooks (OG manual workflows, utils)
│   ├── 13-0_makeScenarioDir.ipynb           ← stores all res data into *.csv
│   ├── 14-5_makeScenarioMapsRobo.ipynb      ← AVALANCHE SCENARIO ROBOTER
│   ├── WIN-14-5_makeScenarioMapsRobo.ipynb  ← AVALANCHE SCENARIO ROBOTER (windows version)
│   ├── ...
│   ├── 90-5_prepForest.ipynb                ← pcc forest tif generation
│   └── ...
```

Running cairos ...

Fill in `local_cairosCfg.ini` → `[MAIN]` with your project info and input filenames (the files must exist in the run's `00_input/` folder once the project is initialized).

From `cairos/` (where `runCairos.py` is):

```
pixi shell --environment dev
pixi run cairos
```

- after first initialzation run. Upload 00_input files and run again...

What the workflow does (Steps 00–12)

Step 00 — Initialize project folders (UPDATE NEEDED)

Creates the standardized structure for your run directory based on cairosCfg.ini `[MAIN]`

- ... `<workDir>/<project>/<ID>/`:

```
├── 00_input/               ← DEM, FOREST, BOUNDARY (user-provided inputs)
├── 01_praDelineation/      ← Step 01 outputs: PRA base raster + aspect
├── 02_praSelection/        ← Step 02: PRA selection results
├── XX_praBottleneckSmoothing/ ← Step 03: smoothed catchments / bottlenecks #not used ATM
├── 04_praSubcatchments/    ← Step 03: subcatchment rasters + polygons
├── 05_praProcessing/       ← Step 04: processed PRA metrics
├── 06_praSegmentation/     ← Step 05: segmented PRA polygons
├── 07_praAssignElevSize/   ← Step 06: assigned elevation/size
├── 08_praPrepForFlowPy/    ← Step 07: prep inputs for FlowPy
├── 09_flowPyBigDataStructure/ ← Step 08: BigData tree (SizeN/scenario leaves)
├── 10_flowPyOutput/        ← Step 10–11: FlowPy outputs / size results
├── 11_avaScenDir/          ← Step 11: scenario directories
├── 12_avaScenMaps/         ← Step 12: scenario maps
├── 91_plots/               ← plots & diagnostics
└── 92_GIS/                 ← GIS-ready exports
```

Log file is written to your run directory:

```
<workDir>/<project>/<ID>/runCairos_YYYYMMDD_HHMMSS.log
```

If you see:

```
...Required input files are missing in ./00_input:
  - DEM=<name>.tif
  - FOREST=<name>.tif
  - BOUNDARY=<name>.shp
```

copy/prepare these files in the run's `00_input/` directory. Their names must match your INI's `[MAIN]` keys.

Steps 01–08 — PRA processing (com1PRA)

Step	Module	main INI sections
01	`com1PRA/praDelineation.py`	`[praDELINEATION]`, `[MAIN]`
02	`com1PRA/praSelection.py`	`[praSELECTION]`
03	`com1PRA/praSubCatchments.py`	`[praSUBCATCHMENTS]`
04	`com1PRA/praProcessing.py`	`[praPROCESSING]`
05	`com1PRA/praSegmentation.py`	`[praSEGMENTATION]`
06	`com1PRA/praAssignElevSize.py`	`[praASSIGNELEV]`
07	`com1PRA/praPrepForFlowPy.py`	`[praPREPFORFLOWPY]`
08	`com1PRA/praMakeBigDataStructure.py`	`[praMAKEBIGDATASTRUCTURE]`

NOTE: table is only showing the main INI section. To ensure the right inputs for the single processing steps other ini parameter are taken in to accout as well!

Output of Step 08 is the **BigData** tree:

```
09_flowPyBigDataStructure/
└── <Root>/                              ← BnCh2_subC500_100_5_sizeF500
    ├── <CaseA>/                         ← e.g. pra030secS-2000-2200-3
    │   ├── Size2/
    │   │   ├── dry/
    │   │   │   ├── Inputs/
    │   │   │   │   ├── REL/pra*.tif     ← Step 09 input (pra*.tif)
    │   │   │   │   ├── ALPHA/alpha.tif  ← Step 09 output / Step 10 input
    │   │   │   │   ├── UMAX/umax.tif    ← Step 09 output / Step 10 input
    │   │   │   │   └── EXP/exp.tif      ← Step 09 output / Step 10 input
    │   │   │   └── Outputs/com4FlowPy/...  ← Step 10 output / Step 11 input
    │   │   └── wet/...
    │   └── Size3/
    │       └── dry/...
    └── <CaseB>/
        └── Size2/...
            └── dry/...
            ...
```

Terms & naming

- **Root** – parameter case folder
  Naming formed by `[praPROCESSING]`, `[praSUBCATCHMENTS]`, `[praSEGMENTATION]`, e.g. `BnCh[minDiagonalNeighborsPass2]_subC[streamThreshold]_[minLength]_[smoothingWindowSize]_sizeF[sizeFilter]`
  - Name from default PRA processing parameter: `BnCh2_subC500_100_5_sizeF500`
- **Case** – one PRA release scenario
  Naming formed by `[praDELINEATION]`, `[praSELECTION]`, `[praASSIGNELEV]`, `[avaPARAMETER]`, e.g. `pra[selectedThreshold]sec[aspectSector]-[elevationBand1]-[elevationBand2]-<max case size>`
  - Where "`max case size`" = **max potential path mobility from CAIROS concept** (defined by `[praSEGMENTATION]`).
- **SizeN** – size class
  Depends on "`max case size`" and `[avaPARAMETER].sizeRange`.
  For `pra...-4` → `Size2`, `Size3`, `Size4`.
  For `pra...-3` → `Size2`, `Size3`.

- **Scenario** – flow regime (`dry/`, `wet/`).
- **Leaf** – `SizeN/scenario/` with `Inputs/` + `Outputs/`.

Steps 09–12 — FlowPy side

Step 09 — Parameterization (per leaf)

- Code: `in2Parameter/compParams.py`
- Inputs: DEM + PRA release (`Inputs/REL/pra*.tif`)
- Uses `[avaPARAMETER]` and `[avaSIZE]` to compute **ALPHA / UMAX / EXP** once per leaf.
- **Folder rule:** if a leaf path contains `.../SizeN/...`, the computed "size" is **clamped** to `N` before mapping to ALPHA/UMAX/EXP.
- DEM selection is handled by `workflowUtils.demForLeaf(...)`:
  - BigData leaves (default): use `00_input/<DEM>` from `[MAIN].DEM`
  - Single / non-BigData run: fall back to the leaf's `Inputs/DEM.tif` (if present)

Step 10 — Run FlowPy (per leaf)

> Update: FlowPy is now called directly via AvaFrame — no `runCairosFlowPy.py` anymore.

- Driver: `cairos/runCairos.py`
- FlowPy INI: `AvaFrame/avaframe/com4FlowPy/com4FlowPyCfg.ini`
  - Copy to `local_com4FlowPyCfg.ini` and edit that version
- Example important settings for CAIROS runs:

```
[GENERAL]
infra = False
previewMode = False
variableUmaxLim = True              #important for Cairos
varUmaxParameter = uMax             #important for Cairos
variableAlpha = True                #important for Cairos
variableExponent = False            #important for Cairos
forest = False                      #important for Cairos
...

#computational defaults:
tileSize = 15000
tileOverlap = 5000
procPerCPUCore = 1
chunkSize = 50
maxChunks = 500

[PATHS]
outputFileFormat = .tif
outputNoDataValue = -9999
outputFiles = zDelta|travelLengthMax|fpTravelAngleMax|relVolMin|relVolMax|relId
useCustomPaths = False
deleteTempFolder = True
useCustomPathDEM = True            #important for Cairos
workDir =
demPath = .../10DTM_pilotSella.tif #important for Cairos
...

[FLAGS]
plotPath = False
plotProfile = False
saveProfile = True
writeRes = True
fullOut = False
```

Step 11 — Back-map FlowPy outputs to “size” (optional)

- Code: `in2Parameter/compParams.py::computeAndSaveSize`
- Controlled by `[WORKFLOW].flowPyOutputToSize`
- Writes to `<leaf>/Outputs/com4FlowPy/sizeFiles/res_<uid>/...`
    - `<uid>` comes from avaframe

Step 12 — Output management (optional)

- Compression: `in1Utils/dataUtils.py::tifCompress`
- Temp cleanup: `in1Utils/dataUtils.py::deleteTempFolder`
- `[WORKFLOW]` flags:
    - `flowPyOutputCompress`, `flowPyDOutputDeleteOGFiles`, `flowPyDeleteTempFolder`

INI: what to set

`[MAIN]`

```
[MAIN]
# where to create the run tree
workDir    = /media/.../cairosTutti
project    = region/subRegion
ID         = testID

# input filenames (must exist in 00_input/)
DEM        = 10DTM_pilotSella_test.tif
FOREST     = 10nDOM_binForestClipExtAgg100_pilotSella_test.tif
BOUNDARY   = AOI_Sella_test.shp

initWorkDir = True
```

`[WORKFLOW]`

Toggle steps:

```
[WORKFLOW]
runAllPRASteps          = True      ; master for Steps 01–08

praDelineation          = True
praSelection            = True
praSubCatchments        = True
praProcessing           = True
praSegmentation         = True
praAssignElevSize       = True
praPrepForFlowPy        = True
praMakeBigDataStructure = True

flowPyInputToSize       = True      ; Step 09
flowPyRun               = True      ; Step 10
flowPyOutputToSize      = True      ; Step 11
flowPyOutputCompress    = True      ; Step 12.1
flowPyDOutputDeleteOGFiles = True   ; Step 12.2
flowPyDeleteTempFolder  = True      ; Step 12.1
```

Flow types & sizes

```
[avaPARAMETER]
flowTypes  = dry,wet     ; scenarios to enumerate
sizeRange  = 2-5         ; or comma list like 2,3,4,5

# Run a single leaf instead of the BigData tree (optional)
useCairosBigDataStructure = True
avaFrameDir =            ; leave empty to enumerate BigData

cpuCount = 8
saveEffectiveConfig = True
```

```
[avaSIZE]
; mapping rules (examples)
sizeMax = 5
alphaSize2 = 34
deltaAlpha = 3
uMaxSize2 = 10
deltaUMax = 17
; EXP often constant for a run; Step 09 writes exp.tif accordingly
```

Logging

- Console logging is configured in `runCairos.py` and **augmented** with a `FileHandler` attached **after Step 00**, writing to:

  ```
  <workDir>/<project>/<ID>/runCairos_YYYYMMDD_HHMMSS.log
  ```

- This captures INFO/WARN/ERROR from **all modules** (root logger handler).
- You'll still see logs in the terminal; the same lines go into the file. To only keep file logs, remove/disable the default `StreamHandler` setup in `__main__`.

Running a single leaf

Instead of enumerating the BigData tree, set in `[avaPARAMETER]`:

```
useCairosBigDataStructure = False
avaFrameDir = /absolute/path/to/.../SizeN/dry
```

- Step 09 will parameterize **that** leaf.
- Step 10 will run FlowPy for **that** leaf only.

---

thats it for now - tbc...
