IDE4EEG

(integrated development environment for electroencephalographic data analysis)

User-friendly Python package with graphical interface
for design and debugging of EEG analysis pipelines

User Manual

Version 0.9

5 September 2026

Written by Piotr Durka

Faculty of Physics, University of Warsaw

with the assistance of Anthropic's Claude (model: claude-opus-4-8),
in cooperation with the IDE4EEG developers:

Anna Duszyk-Bogorodzka, Piotr Biegański,
Monika Tutaj and Andrzej Oreszczuk,

who reviewed and approved technical content.

Contents

Conventions and MNE integration

IDE4EEG is built on top of MNE-Python: every loaded recording lives in memory as an `mne.io.Raw` object, every cut signal as `mne.Epochs`, and the canonical save format is MNE-FIF (`*-raw.fif` for continuous data, `*-epo.fif` for epochs). Channel types use MNE's vocabulary (`eeg`, `eog`, `emg`, `ecg`, `bio`, `stim`, `misc`, …); montages are taken from MNE's built-in library (`standard_1020`, `standard_1005`, `biosemi64`, …). When in doubt about a parameter's meaning, the corresponding MNE function's docstring is the authoritative reference—IDE4EEG forwards most parameters unchanged.

SI units and the V vs µV convention

Volts internally, microvolts at every user-facing edge. MNE-Python stores voltages in **volts (V)**—not microvolts—and IDE4EEG keeps them in volts everywhere *inside* the pipeline. But every human-facing boundary—the values you type, the parameter tables in this manual, the GUI tooltips, plot axes, and review windows—is in **microvolts (µV)**, in concordance with the electroencephalographic convention. Frequencies and times pass straight through in MNE's native units, unconverted: `filters.highpass_freq = 0.5` means **0.5 Hz**; epoch offsets, window lengths and mark durations are in **seconds** (or **milliseconds** where a field says `ms`).

Voltage is the only quantity that gets converted. A config value expressed in microvolts carries a **`_uv` / `_uV` suffix**, and IDE4EEG multiplies it by `1e-6` to reach MNE's volts before use:

- ICA rejection—`reject_uv = 500` (500 µV).
- EEG-artifact amplitude anchors—`min_amplitude_uv`, `slope_min_amplitude_uv`, `p2p_ceiling_uv` (µV).
- Bad-channel flat floor—`choosing_channels.prep.flat_sd_threshold_uV` (µV). The correlation and robust-deviation criteria are dimensionless robust z-scores, so they carry no unit suffix.

On the way out the reverse conversion applies: plot axes and the review browser show `data × 1e6` so you read µV, and derived quantities (band power in μV^2, MP-atom amplitudes in µV) are reported in microvolts at the boundary even though they are computed in volts. Every parameter table in this manual names the unit it expects, and the GUI tooltips repeat the friendly unit (µV, ms) regardless of how the value is stored on disk.

Number entry—decimal separator and sign

IDE4EEG uses **. (period) as the decimal separator everywhere**, regardless of the system locale. This matches TOML's number grammar (`config.toml` floats always use `.`), and Python's `float()` (the codebase parses every text field with `float(text)`).

Where MNE code is invoked directly

Several steps in the **Preprocess** and **Analysis** panels wrap MNE functions transparently: IDE4EEG supplies the parameters, MNE does the work.

Preprocess

- Resampling—`raw.resample(sfreq)`.
- Filtering (FIR mode)—`raw.filter(...)`. IIR mode uses `scipy.signal.iirdesign` + `sosfiltfilt`.
- Montage and re-reference—`raw.set_montage(...)`, `mne.set_eeg_reference(...)`.
- ICA—`mne.preprocessing.ICA` (Picard backend), with component labelling through `mne_icalabel` (ICLabel) or `find_bads_eog` / `find_bads_ecg` / `find_bads_muscle`.
- Cutting and saving segments—`mne.find_events` / `mne.events_from_annotations` for segment boundaries, `mne.Epochs` for cutting, and FIF I/O for the `-epo.fif` output.

Analysis

Methods grouped under MNE-wrapped analyses (4.4–4.9: ERP, Spectra, Time-frequency, Spatial, Comparison, Inverse solutions), are thin wrappers over the matching MNE functions—six collapsible category panels.

Inverse solutions. All four entries in the **Inverse solutions** panel (4.9) call MNE inverse-solution routines: the classical ERP dipole fit (`mne.fit_dipole`), the minimum-norm family (`mne.minimum_norm.apply_inverse`, method ∈ MNE / dSPM / sLORETA / eLORETA), the LCMV beamformer (`mne.beamformer.make_lcmv` + `apply_lcmv`), and MxNE / iRMxNE (`mne.inverse_sparse.mixed_norm`). MNE is also the inverse solver behind **MMP → dipole sources** (4.1): the time-frequency MP decomposition itself is IDE4EEG-native (`empi`), but turning each MP atom into a current source is done by `mne.fit_dipole` against an MNE-derived BEM and forward solution.

Independent of MNE

Matching Pursuit-related functionalities (MP decomposition, MP filter, MMP-> Dipole sources, EEG profiles) are based upon the scripts of the Warsaw group used in previous publications. Connectivity code (MVAR / DTF / PDC connectivity solver) is adopted (with some fixes) from https://github.com/dokato/connectivipy. Gaze detection pipeline (Detect gaze artifacts) was designed from the scratch, using L2CS-Net + InsightFace libraries. Similarly, implementations of the algorithms for detection of bad channels and EEG artifacts were designed based upon dedicated, LLM-assisted literature reviews: artifact-detector review, artifact-marking review and segment-drop aggregation review. Outputs of these functions are wrapped into MNE-compatible structures where useful, sources are quoted in descriptions of respective functions.

---

1. Config tab

The Config tab holds settings that apply to the whole pipeline: bundled helper applications, control how many CPU cores it uses, switch global behaviours on or off, and load/save TOML configuration files.

Loading and saving config files

The bottom row of the tab has four buttons (also reachable via the **File** menu):

- **Load pipeline settings .toml** (Ctrl+L)—read a `.toml` file into the GUI. Every editable widget across all tabs is repopulated.
- **Save pipeline settings .toml** (Ctrl+S)—write the current GUI state out as `.toml`. The format is documented in Appendix C. **Portable bundles:** if your data (signal, MP book, reference faces) sits inside the folder you save into, IDE4EEG offers to write those paths **relative** so the config + data travel together to another machine or install; data outside that folder stays absolute. Choose **Relative (portable)** to make a shareable bundle, or **Absolute** for machine-local paths. A portable config loads correctly from any working directory (GUI *or* `ide4eeg --run`). Only *data* paths are affected—**no helper tool path is ever written to the config** (Svarog, empi, ConnectiVIS and fsaverage are re-found from `~/.obci/` at each launch), so a saved config is already portable in that respect whichever choice you make. Every save (portable or not) additionally **drops the auto-derived output directories** (`output_path`, `preprocessing_output_path`, `analysis_output_path`) whenever they still equal the values IDE4EEG computes from the input file: they are re-derived from the input at load time, so writing them would pin one machine's folder layout—and the name of the recording they were derived from—into the config. A path you set by hand is kept verbatim.

Only what you changed is written. A saved `.toml` contains just the settings that differ from this version's built-in defaults; everything else is refilled on load. That keeps configs short and readable, and it means a config stays valid across upgrades—a key you never set simply follows the new default. Two consequences worth knowing:

- **Minimising works per *section*, not per key.** Once any value in a section differs, the *whole* section is written, defaults included—so a section can list keys you never touched alongside the one you did. That is cosmetic, not a bug. (For example, flipping one detector toggle writes the whole `[eeg_artifacts]` section—every threshold and enable flag—not just the key you changed.)
- **Paths are always written with forward slashes** (`C:/Users/you/rec.raw`), on every platform. You never have to *type* them that way—Windows accepts `\` and `/` alike, and the **Browse** buttons keep working exactly as before—but a saved config stores the `/` form, so the Input-tab fields will show it after you reload. This is deliberate: a config written on Windows then opens unchanged on macOS or Linux, a shared config stops producing a spurious diff every time a collaborator on the other platform saves it, and a path can never be mistaken for a text escape (`C:\Users\...` contains `\U`, which some readers take as the start of a unicode escape rather than a folder name). Nothing about how the pipeline finds your files changes.
- **A stale config cleans itself.** Loading and re-saving an old `.toml` drops keys this version has removed (each is reported once in the log as it goes). If a config has accumulated cruft from earlier versions, open it and save it.

  Starting clean. A config saved from a long GUI session can accumulate settings for analyses you are no longer running—channel selections in particular, since a loaded recording makes every channel panel write out its current state. To get a minimal config for one specific analysis, use **File → Reset → Pipeline settings to defaults**, set up just that analysis, and save. Everything you did not touch is then omitted rather than recorded. - **Export Python script...**—write a self-contained Python script reproducing the current pipeline. The generated `.py` contains only the non-default parameters and runs independently of the GUI—useful for batch processing, automation, cluster submission, or sharing reproducible analyses. The same **portable** prompt is offered: a portable script anchors its data paths to the script's own folder (a runtime `_here()` helper), so the script + its data run from anywhere. Exported scripts are pure `f(input, config)` and never open an interactive review; they run **fully automatic**. Manual review decisions you made in the GUI are *not* carried into the script—they live in the GUI session (per-recording), not in the config the script embeds. See Appendix F for a side-by-side comparison with `config.toml`. - **Reset Defaults**—restores the shipped **pipeline config only** (after a confirm dialog); it leaves your saved preferences and recent-files list untouched.

If a `config.toml` exists in the working directory when IDE4EEG launches it is loaded automatically. Otherwise the GUI starts with sensible defaults—no config file is required.

External tool paths

These paths are auto-detected at startup. On **first launch** of a fresh install, IDE4EEG silently **downloads** the missing helpers (SVAROG with bundled empi, ConnectiVIS) into `~/.obci`—they are treated as crucial parts of the package, not optional add-ons. A modal progress dialog appears while the ~80–150 MB total downloads in the background; click **Cancel** to defer and use the Config tab's per-row **Download recent** button later. The Adoptium Temurin JDK is handled differently: it is not silently downloaded but installed through the OS package manager behind a consent prompt—Homebrew (`brew install --cask temurin`, latest LTS) on macOS, winget (`EclipseAdoptium.Temurin.17.JDK`) on Windows; on Linux IDE4EEG only prints a `sudo apt install openjdk-17-jdk` command for you to run yourself. CLI / batch mode (`ide4eeg --run config.toml`) downloads the `~/.obci` helpers silently, but the Java runtime still goes through the package-manager (macOS/Windows) or printed-command (Linux) path.

The per-row **Download recent** button on the Config tab forces a refresh—it downloads the latest GitLab CI artifact and overwrites the existing install. Use this when you want to update to a newer build; the prompt explicitly warns when other helpers will also be re-fetched.

See Chapter 8 for what each helper does.

Parameter	Auto-install path	Description
`svarog_jar`	`~/.obci/svarog/svarog-standalone-*.jar`	Svarog standalone JAR (signal viewing, MP book display, interactive review).
`connectivis_jar`	`~/.obci/connectivis/connectivis.jar`	ConnectiVIS JAR (3D viewer for dipole sources and connectivity arrows).
`empi_path`	`~/.obci/svarog/empi-*-<platform>`	empi binary for Matching Pursuit decomposition.

Download recent drops files at the paths above—the `*` is a version stamp; `<platform>` is `macos-x64` / `macos-arm64` / `linux-x64` / `windows-x64`. Each row is a **read-only status display**: it shows what auto-detection found at startup, or *not found* with the folder to put the file in. None of these paths is a config setting—a tool is discovered, never configured—so to use a custom build, place it at the path the row names (the `SVAROG_JAR`, `EMPI_PATH`, and `CONNECTIVIS_JAR` environment variables are the exception—an override for a jar or binary kept elsewhere).

Video backends for Svarog video sync. Svarog plays video frames synchronized with EEG signal scrolling using one of three backends, in priority order:

1. **mpv**—bundled with the standalone ZIPs and declared as `Depends:` in the `.deb`. The **Download recent** button extracts portable mpv into `~/.obci/svarog/mpv/` on Linux and Windows from the official mpv release archives. Plays MKV / MP4 / AVI / WebM directly via libavcodec.
2. **JavaFX 17.0.15**—embedded fallback. Built into the *full* Svarog standalone JAR; the default *light* JAR excludes the native JavaFX libraries to keep the download small. **Codec coverage is narrow: MP4/H.264/AAC, FLV, FXM, M3U8 only—JavaFX cannot play MKV, WebM, AVI, or MOV containers**, so this fallback only covers studies recorded as MP4. Recordings from OBS, ffmpeg-piped webcams, or any pipeline producing MKV need mpv or VLC.
3. **VLC**—opt-in for users with an existing system VLC install. No download needed; Svarog detects and uses it automatically when neither mpv nor JavaFX is present. Plays the same broad container set as mpv (MKV/MP4/AVI/WebM/MOV), so it's the practical workaround for MKV recordings on macOS without Homebrew.

On **macOS**, portable mpv is impractical for the pip-installed app (it would require bundling ~30 dylibs from `/opt/homebrew/lib` and rewriting their rpaths). Instead, IDE4EEG checks whether Homebrew is installed; if so, the **Download recent** button asks for explicit consent and runs `brew install mpv` on your behalf. If you decline or Homebrew is missing, the button still installs Svarog/empi/ConnectiVIS successfully and reports that mpv is unavailable—install manually with `brew install mpv` to enable video sync. Note that without mpv on macOS, JavaFX becomes the only fallback, which means video sync works **only for MP4/H.264 recordings**; MKV/WebM/AVI files won't play. For non-MP4 study recordings, install either mpv or VLC before launching Svarog.

The macOS **full .dmg** sidesteps all of this: it bundles a self-contained arm64 mpv (from the stolendata.net portable build) and a static arm64 ffmpeg (from osxexperts.net) inside `IDE4EEG-full.app/Contents/Resources/`, and a PATH prepend in `ide4eeg/__init__.py` makes them discoverable to SVAROG's `Runtime.exec("mpv")` and `Runtime.exec("ffmpeg")` calls.

L2CS-Net (default eye-gaze backend). A separate **Download** button at the bottom of the Tool Paths group installs PyTorch + the `l2cs` Python package (with `face-detection`, which `l2cs` imports) + the Gaze360 weights (~96 MB checkpoint, MIT-mirrored from Ahmednull/L2CS-Net) in one shot. Total install ~500 MB—the CPU PyTorch wheel, which is all L2CS gaze inference uses; the installer requests the CPU build on every platform (on Linux a bare `pip install torch` would otherwise pull a ~2 GB CUDA build that IDE4EEG never runs). L2CS provides true per-eye gaze direction; without it, the only available backend is the InsightFace head-pose fallback, which measures where the **head** is pointing rather than where the **eyes** are looking (see §3.10 Gaze for the full comparison). The two git packages are installed with `--no-deps` on purpose: `l2cs` declares the non-headless `opencv-python`, which would overwrite the OpenCV the video stack installed (see the conflict note below); every other dependency it declares is already present by that point. The installer verifies `import l2cs` afterwards and reports what is missing if upstream has since added one.

Intel Mac: this button is the one case where the install *succeeding* is the problem—IDE4EEG warns you first and lets you cancel. See **Intel Mac + PyTorch** in Known wheel gaps for why, and use the InsightFace head-pose backend instead.

Example datasets panel. Below Tool Paths, a panel offers the bundled example datasets (P300 + video, sleep spindles, sleep EEG profile) as one-click downloads into `~/IDE4EEG_examples/`. Each download records which package version it installed, so IDE4EEG can tell a current copy from one fetched before a release changed the data or its config. When a newer package is pinned, that example's row shows ⚠ **superseded** with its own **Update** button, and the panel's main button becomes **Download / update examples (N)**. Updating re-downloads only the examples that need it.

This matters for reproducibility: the 0.9 P300 package corrected event onsets that were 100 ms early in earlier releases, and replaced four configs with a single one. A copy downloaded before that produces ERPs that will not match the documented numbers. Examples installed by a version of IDE4EEG that predates version recording are reported as superseded, because their version cannot be established—updating once records it. Once every example matches its pin, the button returns to **Delete examples**.

Video processing panel. A persistent panel under Tool Paths reports the live status of the five core Python packages the facetag pipeline needs: OpenCV, PyAV, imageio, InsightFace, ONNX Runtime. (PyTorch + L2CS-Net live in the dedicated **L2CS-Net** row above—they have their own one-shot installer that also fetches the Gaze360 weights file.) Each package row shows one of:

- `[ok]` (green)—installed and importable.
- `[--]` (amber)—missing, but a clean `pip install` is expected to succeed on this platform.
- `[!!]` (red)—missing and platform-blocked: the wheel-availability or build-tooling situation on this (Python, OS, arch) cell prevents a clean install. The blocker reason is shown inline below the affected row. One cell is seeded today:
  - **Intel Mac + Python 3.14**—`onnxruntime` has no cp314 wheel for `darwin x86_64` (upstream dropped x86_64 macOS entirely after 1.23.2). Recommended: install Python 3.13 via Homebrew (`brew install python@3.13`) and recreate the venv. Apple Silicon Macs have working wheels and don't need this workaround.

  A second cell, **Linux + Python 3.14**, was seeded until 2026-07-31: `insightface` and the transitive `stringzilla` had no cp314 wheels, so pip built them from sdist and the row demanded a compiler plus `python3.14-dev` headers. Upstream has since shipped—`insightface` 1.0.1 is a pure `py3-none-any` wheel and `stringzilla` / `simsimd` publish cp314 manylinux wheels for x86_64 and aarch64—so Linux installs clean on 3.14 and the entry was removed.

A cv2 row can also report **conflicting OpenCV installs**. IDE4EEG uses exactly one OpenCV distribution, `opencv-python-headless`. All four OpenCV distributions on PyPI (`opencv-python`, `opencv-contrib-python`, and their `-headless` builds) ship the same `cv2/` files, and pip resolves by project name, so it cannot see that installing a second one overwrites the first: `import cv2` then gives whichever wheel was unpacked last, no matter which one you asked for. The headless build matters beyond disk size—on Linux a non-headless OpenCV points `QT_QPA_PLATFORM_PLUGIN_PATH` at its own bundled Qt 5.15 plugins the moment `cv2` is imported, which can abort IDE4EEG's Qt6 interface with "Could not load the Qt platform plugin xcb", and it needs `libGL.so.1`, absent on headless machines.

Click **Fix** on that row and IDE4EEG repairs it for you: it uninstalls *every* OpenCV variant and reinstalls the one it needs. Removing them all first is required rather than tidy—because they share files, uninstalling one deletes them for all, and the reinstall is what puts a single consistent copy back. `import cv2` therefore stops working for the few seconds between the two steps. The in-app **Install missing** button runs the same repair automatically at the end of an install.

This is worth knowing if you install from the command line: `pip install ide4eeg[video]` ends in this state **by design of the packages involved, not by mistake**. `insightface` requires the non-headless `opencv-python`, and a pip extra has no way to say "install this package but not its OpenCV". So after a manual `pip install ide4eeg[video]`, run the repair once:

```
python -m ide4eeg.install_runner --repair-opencv
```

It prints what it removes and reinstalls, exits 0 when the install is already unambiguous (so it is safe to re-run), and exits non-zero if the repair did not settle. A batch run (`ide4eeg --run …`) with video artifacts enabled also warns when it detects this state and names the command—it warns rather than refuses, because the run itself still works.

The same applies if you pip-installed another OpenCV variant for your own work, or another tool pulled one in. (After the repair, `pip check` reports insightface's `opencv-python` requirement as unsatisfied. That is expected and harmless—`import cv2` gives the headless build, which is what every part of IDE4EEG needs.)

The panel refreshes automatically when the Config tab gets focus, so packages installed in another terminal flip from `[--]` to `[ok]` without restarting IDE4EEG. When any package is missing, an **Install missing** button appears next to the summary; click it to confirm + run `pip install` for the core packages with progress streamed live into a monospace log dialog. After install, packages requiring a process restart (`cv2`, `av`, `insightface`, `onnxruntime`) trigger a "Restart IDE4EEG?" prompt that re-execs the Python process so the new modules load cleanly.

The same diagnostic is available without launching the GUI:

```
python -m ide4eeg.install_diagnostics    # print status only
python -m ide4eeg.install_runner -y      # install everything missing
python -m ide4eeg.install_runner -y --no-l2cs   # skip L2CS extras
```

The CLI runner (`ide4eeg --run config.toml`) runs the same preflight automatically. If a config has `prepare_video_artifacts = true` and any required package is missing or platform-blocked, the run aborts with the diagnostic text before any preprocessing starts.

First-time download troubleshooting (TLS certificate errors). If **Download recent** fails with a TLS error, IDE4EEG handles it automatically in two steps:

1. **First attempt**—validate against certifi's bundled root CA list. Works for almost everyone; `certifi` is in `requirements.txt`.
2. **Fallback attempt**—if certifi fails (e.g. corporate networks with TLS-inspecting proxies), retry against the platform's default CA store.

If both fail, the error dialog shows platform-specific solution:

- **macOS:** run `/Applications/Python 3.XX/Install Certificates.command` once.
- **Linux:** install the `ca-certificates` package, then `pip install --upgrade certifi`.
- **Windows:** `pip install --upgrade certifi`.
- **Corporate networks:** install your IT department's root CA into the system store, or set `SSL_CERT_FILE` / `REQUESTS_CA_BUNDLE` to a corporate bundle.

Where IDE4EEG stores files

A handful of top-level filesystem locations cover everything IDE4EEG installs, downloads, or saves at runtime:

Path	Contents	Typical size
`~/.obci/svarog/svarog-standalone-*.jar`	Svarog standalone JAR (signal + book + tag review).	~50 MB
`~/.obci/svarog/empi-*-<platform>`	empi binary (MP decomposition).	~2 MB
`~/.obci/svarog/mpv/mpv-*-<platform>/`	Portable mpv (Linux / Windows only—macOS uses `brew install mpv`).	~30 MB
`~/.obci/connectivis/connectivis.jar`	ConnectiVIS JAR (3D dipole / connectivity viewer).	~35 MB
`~/.obci/ide4eeg/fsaverage/`	FreeSurfer's average brain model—curated subset (BEM solution + transform, cortical/head surfaces, atlas), downloaded on demand for dipole/source localization & ConnectiVIS 3D (§4.1.1).	~291 MB
`~/.obci/ide4eeg/models/L2CSNet_gaze360.pkl`	L2CS-Net gaze checkpoint (downloaded on first L2CS use).	~96 MB

Path	Contents	Typical size
`~/.obci/ide4eeg/insightface/models/buffalo_l/`	InsightFace face detection / identity models (downloaded on first use).	~200 MB
`~/mne_data/MNE-fsaverage-data/`	Full fsaverage from MNE's OSF mirror—used only as the **last-resort fallback** when the managed `~/.obci/ide4eeg/fsaverage/` subset is absent and cannot be downloaded (§4.1.1). MNE-managed cache, not bundled.	~0.7 GB
`~/.obci/ide4eeg/stage5_dismissed.json`	Per-rule "don't show again" dismissals (created on first dismissal).	< 1 KB
`~/.obci/ide4eeg/logs/ide4eeg.log`	Rotating diagnostic log—the same messages the Run-tab log panel shows, timestamped and saved to disk (keeps 3 rotated files). Survives across sessions; attach it to a bug report.	≤ ~8 MB
`~/IDE4EEG_examples/<name>/`	Example datasets + their pipeline outputs (each example is self-contained).	~50–200 MB per example
`<venv>/lib/python3.X/site-packages/ {cv2,av,imageio, insightface,onnxruntime, torch,torchvision,l2cs}/`	Python video stack—installed via `pip` into the active venv.	~250 MB core; +~500 MB for the CPU PyTorch + L2CS gaze stack

These principles drive the layout:

- `~/.obci/` is the OBCI ecosystem convention shared with Svarog and other OBCI tools. External binaries (Svarog, empi, mpv, ConnectiVIS) live in subdirectories chosen by Svarog itself—Svarog's resolver probes `~/.obci/svarog/mpv/...`, so we install matching that path. We do not move these.
- `~/.obci/ide4eeg/` holds runtime caches IDE4EEG manages directly: model weights, dismissal state, and per-user data the user shouldn't have to back up. Python packages can't live here—pip places them in the active venv's `site-packages/`, which is the only third location.
- `~/IDE4EEG_examples/` is a plain user-visible folder under `$HOME`, deliberately *outside* `~/.obci/`. Reason: when you run the pipeline on `~/IDE4EEG_examples/dipole_spindles/dipoles_example.raw` and don't override `output_path`, IDE4EEG writes results into the parent of the input file (`~/IDE4EEG_examples/dipole_spindles/IDE4EEG_OUT_*/`).
- `~/mne_data/` is MNE-Python's *own* dataset cache, not managed by IDE4EEG. IDE4EEG normally serves dipole / source-localization from the curated subset it provisions into `~/.obci/ide4eeg/fsaverage/` (§4.1.1); the full fsaverage only lands in `~/mne_data/` as a last-resort fallback, when that managed subset is absent and cannot be downloaded.

To remove IDE4EEG's runtime data:

- *Model weights and InsightFace cache*—`rm -rf ~/.obci/ide4eeg/` frees up to ~300 MB.
- *Python video stack + L2CS bundle + InsightFace cache*—click **Uninstall video tools** in the Video stack section header. It runs `pip uninstall -y` for all OpenCV variants (`opencv-python`, `opencv-contrib-python`, and their `-headless` builds), `av`, `imageio`, `insightface`, `onnxruntime`, `torch`, `torchvision`, `l2cs`, `face-detection`, `gdown` (worker thread, streamed log) and then deletes the L2CS weights file + the InsightFace `buffalo_l/` cache. mpv is preserved (Svarog uses it independently). Frees up to ~700 MB. Equivalent CLI form: `pip uninstall <list>` from inside the venv plus `rm -rf ~/.obci/ide4eeg/{models,insightface}/`.
- *External binaries* (Svarog, empi, mpv, ConnectiVIS)—leave at `~/.obci/svarog/` and `~/.obci/connectivis/` unless you've confirmed no other OBCI tool uses them.

Parallel jobs

The Parallel jobs field controls how many parallel worker processes (via https://joblib.readthedocs.io/) IDE4EEG uses for CPU-heavy parts of the pipeline. This is the same knob MNE and scikit-learn expose internally, unified behind one entry point.

GUI default—auto-fallback. In AUTO mode the field is left **blank** (= AUTO / 0); the resolved value min(physical_cores - 2, available_ram_gb // 2), floored at 1—use all cores except two, capped by RAM at roughly 2 GB per worker—is shown only as a **greyed placeholder**, not written into the field (pre-filling it would bake AUTO into an explicit value and defeat the frozen-Windows/empi clamps). On an 8-physical-core / 16 GB machine the placeholder reads 6. The system-info readout below the field shows the same number plus the hardware breakdown (live-updated as you edit), with warnings (yellow) for risky values like "more than half of physical cores" or "RAM footprint exceeds one third of detected RAM."

Headless / CLI default—same auto value as the GUI. "Headless" means running without the GUI—typically `ide4eeg --run config.toml` over SSH on a remote server, a cluster job, or a CI pipeline. When parallelism.n_jobs is unset (or 0) in the TOML, `ide4eeg --run` resolves it to the same auto value the GUI computes (min(physical_cores − 2, available_ram_gb // 2), floored at 1), so a batch run uses multiple cores by default. Two exceptions force sequential regardless: frozen/packaged builds, and non-frozen Windows with no explicit n_jobs (loky workers can crash there). Set parallelism.n_jobs = 1 in the TOML for a fully sequential run.

Every parallelism-aware step uses the same n_jobs value: MNE preprocessing (filter, notch_filter, resample), MNE catalog analyses (PSD, TFR, cluster permutation tests), connectivity bootstraps, and MP decomposition. MP decomposition has its own override field (matching_pursuit.cpu_workers on the Preprocess tab) for power users on RAM-constrained machines, since empi's C++ workers share memory and cost less RAM than joblib's full-Python workers—but by default the MP field is empty and the inherited value is shown as grey italic placeholder text.

Packaged desktop builds—catalog analyses run single-threaded. On the standalone installers and portable bundles (the "Pythonless" Windows / macOS downloads), the MNE catalog analyses—PSD, TFR, ERD/S, the ERP cluster-permutation tests, and source estimation—always run sequentially regardless of this field. A frozen application has no Python interpreter to re-launch, so when it tries to spawn parallel workers it relaunches its own executable; on Windows that handoff crashes or hangs the app (reported against the cluster-permutation test). Forcing those analyses to one worker on frozen builds sidesteps it. Everything else (preprocessing, connectivity, MP) still honours the field, and a normal `pip` / conda install is unaffected and uses the full value everywhere. If you ever see a packaged build hang or crash at an analysis step, set Parallel jobs = 1 as a safe fallback. For practical guidance on choosing a value and BLAS thread pinning details see Appendix E.5.

Output directory and naming (overview)

Results are written to a folder named `IDE4EEG_OUT_<input_filename>/` with `preprocessing/` and `analysis/` subdirectories. By default the folder lives next to the input signal; the `output_path` field on the Input tab overrides this. **Overwrite output** is on by default, so each full Run overwrites the previous results in place (one folder per recording). If you uncheck it, each full Run instead creates a timestamped subfolder (`IDE4EEG_OUT_<input_filename>/<date_time>_<mode>/`) so earlier results are kept. This timestamping applies to the full Run only—the per-step 👁 eye-button previews always write to the in-place folder regardless of the toggle, because they reuse stable paths to locate each step's snapshot (a timestamped preview folder per click would defeat that lookup). Note: if you set a **custom** Preprocessing- or Analysis-output path on the Output tab, that path is used verbatim and **Overwrite output** has no timestamping effect (you chose the exact location).

> **Cross-run reuse needs overwrite ON.** The expensive cached artifacts—the gaze/video not-looking pass, the MP `.db` book, and the preprocessing step snapshots—are reused across runs only when they're found in the *current* run's output folder. With **Overwrite output off**, every full Run writes to a *new* timestamped folder, so the previous run's artifacts are in a different folder and **each Run recomputes them from scratch** (e.g. the gaze video pass re-runs even though nothing relevant to it changed). Keep **Overwrite output on** if you want re-runs to reuse a prior gaze detection / MP book.

Every run folder also gets a copy of the settings that produced it (`config.toml`), in batch runs as well as GUI runs. Re-running that copy creates a new results folder rather than overwriting the one it came from—as long as **Overwrite output** stays off, which the copy records.

For the full table of output subfolders, file naming conventions, and what triggers each artefact, see Chapter 6 (Output tab).

Other Options

Option	Default	Description
Overwrite output	**on**	Overwrite previous results in place (one folder per recording). Uncheck to create a timestamped subfolder per full Run instead. Applies to the full Run only—eye-button step previews always write in place (they need stable paths to find step snapshots). No effect when a custom Preprocessing/Analysis output path is set.
Allow manual modifications (interactive reviews)	off	Master switch deciding whether the on-demand step reviews (bad-channel / ICA-component / segment) may **change** the result. By default a Run never pauses for review—you open a review yourself with a step's decision button (unless you arm the guided sweep, below). Off (default): that button is a 🔒 **view**—the review opens read-only (inspect the automatic picks, can't edit them), so output stays `f(input, config)`; in batch (`--run` / scripts) reviews are fully suppressed so no window can hang the run. On: the button becomes a ✏️ **pencil**—the review opens editable and your edits are **remembered for this recording in the current session** (RAM), applied to every later Run of the same file until you revert them. They are **not written to the config**—the reproducible record of a reviewed result is the saved cleaned signal. A ✋ **marker** on the step shows a remembered decision is active (see When a manual review resets). The switch itself has **no pipeline effect and is in no cache hash**—flipping it never invalidates your step snapshots or the MP book.
↳ **stop at each user-editable step (guided review)**	off	Dependent sub-toggle (needs *Allow manual modifications* on). When armed, a **Run pauses at each enabled editable step**—bad channels, ICA, segments, gaze—opens its editable review, applies your verdict, and continues; the Run button reads **"Run Pipeline (with stops)"**. Each decision is remembered in the session, so a later plain Run of the same file reproduces the same result **without stopping**. Steps that are off are skipped. This is a one-run interaction preference: it is **not saved to config** and never affects a batch `--run` (which has no review windows). Ideal for eyeballing a bad recording to hand-save channels or data the automatic criteria can't express.
Verbose console logging	**on**	Adds detail to the log: the per-channel bad-channel diagnostic tables, and—when a helper app (Svarog / ConnectiVIS) is launched—its full live output plus a DEBUG log level so a *silent* failure reveals its cause. Also echoes startup progress + the launch command to the terminal. A launch *failure* is reported regardless of this switch.
Allow changing the order of preprocessing steps and adding new filters	off	Reveal ↑/↓ arrows on every reorderable step, plus the add/remove-filter controls (see Step order and constraints). Off by default—the standard pipeline order is correct for most workflows.
Display functions and config names in expanded panels	off	Show a small grey row at the top of each preprocessing panel listing the actual Python function it calls and the TOML config sections it reads. The row lives inside the panel body, so it appears when you **expand** a step (a collapsed Preprocessing tab won't show it). Useful for debugging and matching GUI steps to CLI equivalents.

Option	Default	Description
Manual review timeout (min)	**480** (8 h)	How long an interactive review window (Svarog: bad channels / segments / ICA components) may stay open before IDE4EEG stops waiting, closes it, and continues with the automatic result. It is **not** a limit on how carefully you review. It matters mainly for the guided sweep (**stop at each user-editable step**), which is the only mode in which a review sits *inside* a running pipeline and can therefore wedge it; an on-demand 🔒/✏ review opens after its truncated run has already finished, so a forgotten window there blocks nothing but itself. Raise it for long recordings that take a while to inspect; set **0** for **no limit** (waits until you close Svarog yourself, which forfeits the guard). If the limit is hit, any marks you had **not saved** in Svarog are lost. Applies to all three review windows and to the read-only (🔒) inspection view.

---

2. Input tab

Choose the paths, and verify/set metadata of the input file.

Paths

Type the path or browse by clicking the **Input signal** button. It can point to a single EEG file or a directory; when given a directory, IDE4EEG walks it recursively and treats every supported file as a batch member. The supported file formats and their companion-file requirements are listed in Appendix B.

Recent files offer the last few choices, **Clear** wipes the list. **Video (optional)** is needed for §3.10 Gaze. Auto-detected from the input filename (`recording.raw → recording.mp4`). The **Output root** field controls where `IDE4EEG_OUT_<filename>/` is created. Empty = save next to the input file. Selected file can be previewed via the "View in:" row—a **SVAROG** or **MNE** button.

Signal info

Once a file is selected, IDE4EEG reads its header and shows: **Duration**—total recording length (`h mm:ss`), **Sampling**—sample rate in Hz, **Channels**—channel count (second line breaks it down by MNE type, e.g. `Channels (64): 62 eeg, 1 eog, 1 stim`), **Events**—total number of trigger events found (all codes summed, including any unnamed numeric codes). ⚠ `tag file unreadable` is appended to the readout (the Events count still shows its number) if the companion tag/event file can't be parsed, and **est. RAM**—an estimate of **peak** memory during processing, `≈ 3 × raw-data size` (the factor covers the in-memory copies MNE makes while filtering, referencing, ICA, etc.). That peak estimate is then compared to the machine's free RAM (measured via `psutil` / `sysconf`), and if the estimate exceeds ~80% of it, the readout adds an amber ⚠ `(free RAM: X - consider trimming)`. It's advisory only: nothing is trimmed automatically and no window is proposed—it points the user at the Trim signal step to crop the recording.

Channel selection (keep / drop / right-click to change type)

Collapsible panel (collapsed by default) lists every channel with a checkbox. Checked = keep, unchecked = drop. Use this manual selection to restrict analysis to a subset of channels or to exclude named non-EEG technical channels (Photo, Audio, Sample_Counter, …). Nothing is dropped automatically. The panel writes `choosing_channels.selected_channels` / `choosing_channels.dropped_channels` (see Appendix C).

Use it to restrict analysis to a chosen channel subset, or to exclude named channels you don't want—typically non-EEG technical channels (e.g. Photo, Audio, Sample_Counter). Nothing is dropped automatically; only channels you list in `dropped_channels` (or omit from `selected_channels`) are removed. The final channel set is `selected_channels \ dropped_channels`.

Parameter	Default	Description
`dropped_channels`	`[]`	Channels to exclude entirely (e.g. non-EEG technical channels); empty = drop none.

Parameter	Default	Description
`selected_channels`	`"all"`	Channels to keep. `"all"` keeps everything except those in `dropped_channels`.

Channel **types** (`eeg`/`eog`/`ecg`/…) are also reviewed and overridden here: right-click any channel → **Set type** → to override its MNE type. See User overrides for the type-editing details.

Channel types

Every channel in a loaded recording is assigned an MNE channel type via inference procedure described below in Channel type inference. The Input tab is where this typing is reviewed and, when needed, overridden—every later panel on the Preprocess and Analysis tabs reads these type assignments to populate the adaptive **EEG only / EOG only / ...** filter buttons and to power MNE's `pick_types(eeg=True)` calls.

Channel type inference

Where types come from, in order of authority:

1. **File-format metadata.** For FIF, EEGLAB, and BrainVision files, channel types are read directly from the native file metadata (explicit per-channel type tags, chanlocs structures, header fields). Authoritative—IDE4EEG respects whatever the reader assigned.
2. **Readmanager name heuristic.** IDE4EEG's overlay (step 3) delegates every ambiguous (`eeg`/blank) channel name to `readmanager.chtype_heuristic(name)`. The heuristic recognises (in priority order):

Rule	Example names	Type
Substring `eog`	`EOG Left Horiz`, `VEOG`, `EOG Fp1-M2`	`eog`
Substring `emg`	`EMG Chin1`, `EMG Ant Tibia-0`	`emg`
Substring `ecg` / `ekg`	`ECG ECGI`, `EKG_lead1`	`ecg`
Substring `resp`	`Resp Thermistor`	`resp`
Substring `sao2` / `spo2`	`SaO2 SaO2`, `SpO2`	`bio`
Substring `stim` / `trig` / `marker` / `status` / `sync`, prefix `sti`	`STIM`, `Trigger`, `STI 014`	`stim`
Tokenised 10-05 position lookup	`Fp1`, `C3`, `EEG F3-CLE`, `Fp1-M2`	`eeg`
Fall-through	`Aux1`, `Photo`, `Channel_42`	`misc`

Non-EEG substring rules take priority over the position lookup, so `EOG Fp1-M2` (an EOG reference channel using Fp1/M2 references) is correctly typed as `eog` rather than `eeg`. The 10-05 position check tokenises on whitespace/punctuation before matching, so short position names (`A1`, `C3`, `O1`) don't accidentally match inside unrelated words (`audio1`, `Data1`, `misc3`). Requires readmanager ≥ 1.4.0.

3. **IDE4EEG overlay (universal).** After *every* file reader runs (FIF, EEGLAB, BrainVision, EDF, BDF, BrainTech `.raw`), IDE4EEG applies `ide4eeg.input.input._refine_channel_types` to the loaded signal. It (
   (a) respects any specific non-`eeg` type the reader already assigned (FIF/BrainVision/EEGLAB native types, including a deliberately-`misc` channel) and
   (b) delegates only the ambiguous default-`eeg`/blank channels to the readmanager heuristic of step 2. This is why EDF/BDF files—which MNE blanket-defaults to `eeg` because the format has no channel-type field—get the same name-based refinement as a BrainTech recording.
4. **Unknown channels in standard montages.** When most electrodes have recognisable 10-05 names (a "standard-named" recording, recognised EEG channels in the majority), any reader-`eeg` channel the heuristic still cannot place is **demoted to** `misc` so it does not silently join EEG-only operations (CAR, `pick('eeg')`, PSD, ICA, ICLabel, REST). For numbered / non-standard montages (e.g. BioSemi `A1..A32`, `Ch1..ChN`) such channels are instead **kept as** `eeg`. Either way the demotion/keep is announced in the Run log so you can confirm or correct it on the Input tab (see **User overrides** below).

5. **User overrides**—see below.

User overrides

When the automatic heuristic gets a channel wrong—typically for lab-specific names the substring rules don't recognise (`Heart_sensor`, `Chest_strap`, `Channel_42`)—you can override it via the GUI or the TOML config.

In the GUI: expand the **Channel selection** panel in the Signal Info area (the same panel that keeps/drops channels). It lists every channel with its current MNE type. Right-click any channel → **Set type** → submenu to change it. Changes apply live: the override dict updates, every channel panel on Preprocessing/Analysis tabs refreshes, the per-type filter buttons (EEG only, EOG only, …) rebuild to reflect the new counts, and the Signal Info type-count summary updates. Picking **use auto-inferred** removes an override and restores the auto-inferred type.

In TOML:

```toml
[choosing_channels.type_overrides]
"Heart_sensor" = "ecg"
"Chest_strap"  = "bio"
"Aux_L"        = "eog"
"BadChannel"   = "misc"     # exclude from EEG analysis by name
```

Keys are exact channel names (as they appear in the loaded file); values are any valid MNE channel type (`eeg`, `eog`, `emg`, `ecg`, `bio`, `stim`, `misc`, `ref_meg`, `seeg`, `ecog`, `dbs`, `fnirs`). Overrides always win—they take effect after both the file reader and the name-based heuristic have run. Entries whose channel name isn't in the loaded file are logged and silently skipped.

The Review panel is the single entry point for editing channel types. Per-tab channel panels are read-only for types—they use the type assignments (to power the adaptive filter buttons) but don't let you edit them.

Montage (electrode positions)

The **Electrode layout** combo sets the montage: `native (from file)` (default—use whatever positions the file already carries) or a named standard montage (`10-20`, `10-10`, `10-05`, the BioSemi and GSN-HydroCel (EGI) families, `easycap-M1`, `mgh60`, `mgh70`). It writes `electrodes_layout`. **Show head diagram** renders the current layout as a head plot—channels matched to a montage position are blue, unmatched montage positions gray—so you can confirm the layout before running.

Backend: `_set_montage` (`channels_and_signal.py`). Loads a standard montage (e.g. `standard_1020`, `biosemi64`) from MNE's built-in library and applies it via `signal.set_montage(montage, on_missing="warn")`; unmatched channels keep their existing position (or none) and a warning is logged. It writes only metadata into `signal.info`—no signal samples change.

Native-position keep. When the loaded file already provides 3D digitised positions for every EEG channel (e.g. an MNE sample FIF with `EEG 001…` naming + digitised coords), the standard-montage step is skipped to preserve the file's own coordinates—applying `standard_1020` would silently blank them out by name-matching. The config records this with the sentinel value `electrodes_layout = "native (from file)"` (the literal string is exposed as `ide4eeg.NATIVE_POSITIONS_SENTINEL`).

Positionless files. When a file carries no usable channel positions and the layout is left on the `native (from file)` sentinel (the default), there are no positions to keep, so the montage step falls back to applying `standard_1020` (logged at INFO—a clean, expected fallback, not a warning). The Input-tab combo reflects this by showing `10-20` for such a file, so the head diagram, montage-match summary, and generated config all match what the pipeline will run.

Parameter	Default	Description
`electrodes_layout`	`"native (from file)"`	MNE montage name (`"standard_1020"`, `"standard_1005"`, `"biosemi64"`, …) or the literal sentinel `"native (from file)"` to keep the file's own positions. The default is the sentinel: a file that carries its own positions keeps them; a file without positions falls back to `standard_1020`. See the MNE montage docs.

3. Preprocess tab

The Preprocess tab is where you configure the signal-cleaning pipeline and parametrization. Each step has its own collapsible panel with an enable checkbox, parameter fields, and a 💾 Save toggle controlling whether the step's intermediate output is written to disk. It has three sections:

1. **Segmentation setup**—fixed and always runs first. One panel, where you choose between event-locked epochs and timed-window (rest) mode and pick which events to use.
2. **Reorderable steps**—ten steps you can enable individually and, if **Allow changing the order of preprocessing steps and adding new filters** is checked in the Config panel, reorder via ↑/↓ arrows on each panel header. MP decomposition does not modify the signal—it produces the `.db` atom book the Analysis-tab analyses consume; see §3.8 / §3.7.
3. **Final segment handling**—after the reorderable steps, one fixed panel, **Mark detected artifacts**, conditions the artifact marks and optionally drops bad segments. Its rejections can be reviewed by hand, but never during a Run: you open the review on demand from the panel's decision button (§3.12). The pipeline then automatically cuts the signal into epochs and saves the cleaned result—those steps have no panels of their own (see Automatic postamble outputs).

The reorder controls are hidden by default. Tick **Allow changing the order of preprocessing steps and adding new filters** in the Config tab (see Other Options) to reveal them. The default order is correct for most workflows.

> *Internally the pipeline has three phases—Segmentation setup (always runs first), the reorderable steps (user-controlled order), and the postamble (the Mark detected artifacts panel, then the automatic cut / save). In code comments and maintainer docs these are called the "preamble", "reorderable steps", and "postamble".*

GUI icons at a glance

Each step panel's header carries a small cluster of icons (left of the status text). Here is what each means; the detail follows in the sections below. Not every icon appears on every step—**Detect bad channels** and **Detect gaze artifacts** have no 👁, and only review-capable steps (bad channels, ICA, epochs, gaze) show ✏/🔒/✋.

Icon	Meaning
👁	**View** this step's before/after signal (opens Svarog or MNE). Read-only—no pipeline effect.
✏	**Edit** this step's manual review—opens the review (bad channels / ICA / epochs / gaze). Shown when **Allow manual modifications** (Config tab) is on.
🔒	The same decision button when **Allow manual modifications** is **off**—opens the review **read-only**.
✋	A **remembered manual review decision** is active on this step and will be applied on the next Run. Click it to review, amend, or **revert to automatic**. Three states: **lit** = active (applies on the next Run); **faded** = dormant (the step is disabled; it returns when re-enabled); **absent** = no decision at all. The absence is the guarantee: **no ✋ on a step means that step ran, and will run, purely automatically.** See When a manual review resets.
💾	**Save** this step's signal snapshot to disk. A free user toggle, defaulting off, on every step that has one.
⚙	This step writes **auxiliary outputs** (plots / audit files / JSON) beside its result.
↑ ↓	Move this step **earlier / later** in the pipeline (shown only when reordering is enabled).
▶	**Run** the pipeline (or, on an Analysis-tab row, that one analysis).
⚠	A pre-flight / consistency **warning**—hover for details (see the Help tab's **Pipeline invariants** panel).

When the decision button is unavailable. A step you have unticked has no decision to make, so its 🔒/✏ button is greyed out. A step that is *ticked* but missing from the pipeline's effective step order—typical of a hand-edited or older TOML whose `step_order` omits it—still shows a live button,

and clicking it gets a **"Enable *<step>* first"** dialog instead of a review. Re-tick the step on this tab (which puts it back in the order) and try again.

Save toggles (🗄)

Every preprocessing step that *has* a signal snapshot to save has a 🗄 **file-cabinet toggle** in its header row. Opacity encodes state—bright when save is on, dimmed to ~25% when off. The toggle is always **user-driven** and defaults off: enabling a step does not turn its save on automatically. Clicking the 👁 eye button force-flips the relevant saves on so the snapshot persists.

Three steps have **no** drawer at all, because they produce no signal snapshot: **MP Decomposition** (the `.db` atom book is its output, always written when the step runs) and the two marking detectors, **EEG Artifacts** and **Gaze** (their `-tag.txt` marks file is their output, also written unconditionally). For these, the step's own enable checkbox is the whole story.

Saving is what makes re-runs cheap. The drawer is not just housekeeping: a snapshot that was never written cannot be reused. With every drawer off—the factory default—each Run recomputes the whole preprocessing chain from the input file, however little you changed. (The MP atom book is the exception; it has its own reuse check, see MP book reuse.) Turn the drawer on for the expensive steps you are *not* currently tuning—ICA is the usual candidate—and the next Run starts from the latest snapshot still matching your config instead of recomputing it. Hash-based caching (overview) explains how that starting point is chosen.

What gets saved per step:

- **Modifying steps** (Trim, Resample, Reference, Filtering, ICA, MP Filter): the transformed continuous signal as `<base>-<suffix>-raw.fif` in `preprocessing/saved_steps/`.
- **Detect bad channels** writes a `<base>-bad-marked-raw.fif` snapshot carrying the updated `info[bads]` (so the 🔒/✏ decision-button view works—this step has no eye), plus its per-criterion detection list.
- **Marking steps** (EEG Artifacts, Gaze): write a channel-scoped `<base>-tag.txt` marks file—their **primary output**, written whenever the detector runs, with no toggle to gate it. They have **no `-raw.fif` snapshot** (one would just duplicate the previous step's signal); the eye overlays the marks on the preceding step's signal.
- **MP Decomposition**: writes the `.db` atom book whenever the step runs, again with no toggle. Its reuse across runs is governed by the book hash, not by a save flag—see MP book reuse.
- **The cleaned epochs** (the postamble output in `saved_signals/`) have no Save toggle—they are the primary pipeline product and are always saved on a full run.

Clicking the 👁 eye button (below) auto-flips the relevant save toggles (for the steps before), on so the snapshots persist—useful if you wanted to see the before/after and keep it for later too.

The 🗄 drawer represents **snapshots only**. Auxiliary outputs (filter plots, ICA component reports) get their own widgets, typically separate inline checkboxes labelled "Save filter plots", "Save components plot and table", and so on.

View Step Result (👁)

Most preprocessing steps have an **eye button** (👁) in their row header, which opens a synchronised before/after view of the signal around that step.

- **Before** = the nearest enabled preceding step's saved signal (or the original input file, if the step you clicked is the first enabled step).
- **After** = the step's own saved signal.

Clicking 👁 auto-flips the 🗄 save toggles on for this step and its preceding step, so the snapshots persist across future runs.

Freshness / staleness detection. Each saved snapshot FIF carries a hash of the preprocessing config that produced it, embedded in `info["description"]` as `"after <step> [cfg:<hash>]"`. When you click 👁, IDE4EEG compares that hash to the hash of your current GUI config. If they match, the viewer opens directly. If you've changed any preprocessing parameter since the snapshot was written, the hash differs and the snapshot is treated as stale—IDE4EEG auto-runs the pipeline to regenerate it before opening the viewer.

The same hash drives a **write-skip optimisation**: when the pipeline reaches a step whose target snapshot already exists on disk with a matching `[cfg:<hash>]`, the save is skipped (the bytes would be identical anyway).

Run-to-step—the eye button can drive the pipeline. If the saved files don't exist yet (or are stale), clicking the eye button runs preprocessing up to that step instead of asking you to click Run manually. It runs the normal pipeline—the same one the Run tab uses—just stopped at the step you clicked. The truncation happens on a private copy of your settings: on that copy it keeps only the steps up to and including your target, turns every analysis off, and turns saving on for those steps so each writes its snapshot to disk. Nothing is written back into your config file.

Two things in the GUI *are* changed for you, deliberately and visibly: the 🗄 drawers of the clicked step and of its preceding step are switched on (so the snapshots survive for later runs), and if the clicked step was disabled, its enable checkbox is switched on. Both are ordinary toggles—untick them afterwards if you don't want them. Nothing else in the GUI is touched, and the viewer opens automatically when the pipeline finishes; the before and after snapshots, plus any intermediate ones, stay on disk and are viewable later without re-running.

The eye button is available on eight of the ten reorderable steps: Trim, Resample, Reference, Filtering, ICA, MP Decomposition, EEG Artifacts, and MP Filter. **Detect bad channels** and **Detect gaze artifacts** have no eye—neither alters any samples, so the only thing worth showing is the decision itself, reached via the step's 🔒/✏ decision button instead.

Three of the eight open a step-specific viewer rather than the generic before/after split:

- **Trim** opens the **Preview & Trim** window directly. A split view aligns the two signals by sample index, which would mislabel the cropped segment's time axis as `00:00`.
- **MP Decomposition** opens the **MP Book Viewer** on the `.db` atom book—Svarog with the signal synced if it is installed, the built-in book viewer otherwise. The step produces a book, not a modified signal, so there is no before/after to show. It needs an existing book on disk; run the decomposition first if there is none.
- **EEG Artifacts** overlays its `-tag.txt` marks on the preceding saved step's signal—the trace the detector actually saw. It writes no signal snapshot of its own.

The other five (Resample, Reference, Filtering, ICA, MP Filter) open the standard before/after view described above.

The ⚙ badge

Each preprocessing step's header row can show one compact status badge next to its title—a small ⚙ gear glyph (muted olive)—when the step is configured to write auxiliary outputs beyond its signal snapshot. Hover it for a tooltip describing exactly what's enabled. Three steps surface it:

- **Detect bad channels**—when **Save detection plots** is on (`bad_channels.save_plots`).
- **ICA**—when **Save components plot and table** is on (`ICA_EOG.save_components_audit`): topomap PNGs, the classification CSV, `ica.fif`, and an MNE `Report` HTML.
- **Filters**—when **Save filter plots** is on (`filters[*].plot_filt`), per filter block.

The ⚙ lets you see at a glance which steps will produce extra output, without expanding every panel. It is independent of the 🗄 save drawer to its left, which controls the *signal snapshot* separately.

Hash-based caching (overview)

IDE4EEG uses a Merkle-style hash chain to identify the exact configuration that produced each step's signal snapshot. The `[cfg:<hash>]` marker stamped into every saved FIF lets the GUI detect staleness when you change a parameter, drives the eye-button's freshness check (see **View Step Result** above), and powers the write-skip optimisation when re-running a pipeline whose downstream parameters changed but whose upstream snapshots are still valid.

How a run resumes. Snapshots are not consulted step by step: there is exactly **one resume point per run**. Before the reorderable steps begin, IDE4EEG scans the saved snapshots from the *last* step backwards and stops at the first one whose `[cfg:<hash>]` marker still matches the current configuration. That snapshot is loaded as the starting signal, every step up to and including it is skipped, and everything after it runs normally. If no snapshot matches—or none was ever saved—the pipeline runs in full from the input file.

The two detectors are never skipped. The backward scan stops short of **Detect EEG artifacts** and **Detect gaze artifacts**. Each of these builds its list of marked spans in memory, and a saved `-raw.fif` cannot carry that list back, so the resume point always lands *before* the first detector in your step order—the detectors and every step after them re-run on each Run. In the default order the detectors come last, so this costs nothing; but if you move a detector earlier, everything downstream of it (ICA included) is recomputed every time.

For internals—the per-step fingerprint, what's included in / excluded from each step's hash, and the three invariants that keep the chain stable across re-runs—see Appendix E.6.

3.1 Segmentation

GUI panel: Segmentation (always on—its enable checkbox is ticked and locked)

Pipeline phase: preamble—the event-ID build runs first (before any reorderable step); rest markers are attached in the late preamble, *after* the reorderable steps Segmentation setup prepares the coordinate system used by the rest of the preprocessing chain. It does two things:

1. **Builds the event-ID dictionary**—a pure config operation that maps the user's selected event names (GUI event panel, or `[epochs.tags] selected = [...]` in TOML) to integer IDs, consumed later by `mne.Epochs` / `_cut_segments`.
2. **Attaches synthetic rest markers**—only in continuous mode. `REST` annotations mark the synthetic window onsets via `signal.set_annotations(...)`; the original `STIM` channel is **never touched**. Unlike the event-dict build (a pure pre-loop operation), this runs in the **late preamble—*after* the reorderable steps**—so the onsets land in the final post-trim / post-resample coordinate system (which is why rest windows are trimmed-relative, see Mode: continuous (rest)).

Every preprocessing step preserves the event markers the file reader produced. Rest-mode analyses do not overwrite the original `STIM` channel—synthetic window markers go to `REST` annotations instead, leaving file-derived events intact. A recording with hardware triggers analysed in continuous mode keeps both coordinate systems available in the saved `-epo.fif` outputs. The rule is enforced by construction: no preprocessing step writes to `STIM`. Rest-marker generation is purely metadata (`set_annotations` is lazy—no `load_data()` is forced) and downstream consumers pick which event source to read based on the active segmentation mode (annotations in rest mode via `events_from_annotations`, the STIM channel in event-locked mode via `find_events`). Stim channels are filtered by type (`mne.channel_type(info, i) == "stim"`), not by exact name.

Mode: continuous (rest)

Used when `segmentation.mode = "rest"`.

Parameter	Default	Description
`rest_duration`	`[0, ""]`	`[start, end]` time interval (seconds) of the continuous signal to extract for rest analysis. Empty-string or omitted end = full signal.
`window_length`	`20`	Length (seconds) of each analysis window. The rest segment is divided into non-overlapping windows of this duration.

Rest windows are computed in the **trimmed** signal's coordinate system—`rest.rest_duration` means seconds relative to `t=0` of the post-trim data, matching MNE's convention that `raw.times` restarts at 0 after `crop`.

In rest mode the spectral analyses estimate on the *continuous* signal and omit artifact-marked spans rather than cutting fixed windows—see §4.5 (`[segmentation].reject_by_annotation`, default `true`).

Mode: event-locked (epochs)

Used when `segmentation.mode = "epochs"`.

Parameter	Default	Description
`start_offset`	`-0.3`	Epoch start time relative to the event trigger (seconds). Negative = before the event.
`stop_offset`	`0.7`	Epoch end time relative to the event trigger (seconds).
`epochs_baseline`	`"None"`	Baseline correction: `"None"` to skip, `[start, end]` in seconds (e.g. `[-0.3, 0]`), or `["None", "None"]` for baseline across the entire epoch. Inside the list, individual `"None"` entries mean "use the epoch boundary" (so `["None", 0]` = from epoch start to 0 s—the pre-stimulus period, MNE's default). See the preset table below.

Baseline correction (`epochs_baseline`). Subtracts, from every epoch, the mean signal over a chosen window—removing DC offset and slow drift so post-trigger deflections are read against the pre-stimulus level. The window is `(start, end)` in **seconds relative to the trigger** (`t = 0` = event onset); a `None`/blank edge means the epoch's own start or end (MNE convention). The GUI dropdown presets:

Option	Stored value	Meaning
`None`	`"None"`	no baseline correction
`(None, 0)`	`[None, 0]`	epoch start → trigger—the pre-stimulus period; **MNE's default baseline**
`(start, 0)`	`[None, 0]`	identical to `(None, 0)`
`(0, stop)`	`[0, None]`	trigger → epoch end (post-stimulus; rarely used)
`Custom`	`[start, end]`	your own window, e.g. `[-0.3, 0]` for the 300 ms pre-stimulus interval

Example with custom thresholds:

```
[epochs]
start_offset = -0.2
stop_offset = 0.8
epochs_baseline = [-0.2, 0]
```

Tags—selecting which events to include. In the GUI, events discovered from the file appear as checkboxes. In TOML:

```
[epochs.tags]
selected = ["picture_1", "picture_2", "picture_A"]
```

segments preview

The Segmentation setup panel header carries a button that previews where segments will fall before running the pipeline. It opens Svarog on the **raw input signal** with a marks overlay for the current config (a Format-A `-tag.txt` routed to Svarog's `--marks`):

- Windows render as one uniform light-gray `segment` band, with a thin `boundary` divider at each window edge.
- The file's STIM events are overlaid too, one colour per event type (hover a mark to read its label): in **rest mode** they are context markers read from the raw `STIM` channel; in **epochs mode** they are the epoch-anchor events.

Rest mode places one window every `window_length` seconds from `rest_duration[0]` to `[1]` (or the end of the signal if `whole_signal` is checked), offset by `trim_start` so the windows land correctly on the raw signal even when trim is configured. **Epochs mode** places a `[start_offset, stop_offset]` window around each selected event. The button requires an input signal loaded on the Input tab; it opens the raw signal (no preprocessing applied yet). The light-gray band + pinned per-type event colours need a Svarog jar that implements the `--marks-*` flags; older jars show a flat overlay.

Step order and constraints

The following ten panels are the **reorderable steps**—you can enable each individually and rearrange them via ↑/↓ arrows on the panel headers. Each reorderable step has a checkbox in its panel header. When the order controls are visible (Config tab → **Allow changing the order...**), each step also gets ↑/↓ arrows for repositioning.

You may order the steps however you like. No ordering is forbidden and no reorder can be refused. IDE4EEG does, however, know two orderings that are usually mistakes, and says so—as a "Scientific note" when you make the move, and as a warning in the run log:

Recommended order	Why
`bad_channels` before `ica`	A noisy, flat, or uncorrelated channel tends to dominate one ICA component and corrupt the rest of the decomposition.

Recommended order	Why
`reference` before `filtering`	A specific-channel reference (e.g. linked mastoids, typed `misc`) is subtracted as `data - mean(ref)`. Filtering skips non-EEG channels, so referencing *after* filtering re-injects the reference channels' unfiltered drift and line noise into every channel.

These are advice, not rules: the move goes through, the run proceeds, and you are told why the order is unusual. If you have a reason for it, ignore the note.

> **Note.** Step-ordering advice is fixed in code, not configurable. A config that still carries `[preprocessing] hard_constraints` / `soft_constraints` keys loads fine—the keys are ignored and dropped when you re-save (the log notes it once if you had actually customised a value; a config merely echoing an old shipped default is dropped silently). There is likewise no `filtering → ica` recommendation: ICA high-passes its own fit copy internally (`ICA_EOG.fit_highpass_hz`, §3.9), so the benefit no longer depends on where `filtering` sits in the pipeline.

The current step order is stored as `preprocessing.step_order = [...]`. If you omit `step_order` in a hand-written TOML, a reduced legacy order is *derived*—not the canonical default: `trim, resample, bad_channels, filtering`, with `ica` / `gaze_artifacts` / `mp_filter` appended only if the matching legacy `prepare_*` flag is set. It drops `reference`, `mp_decomposition`, and `eeg_artifacts` entirely (a bare config must list `eeg_artifacts` explicitly to run it); `montage` and `chosen_channels` are fixed signal-setup steps handled outside this derived list. To get the full canonical pipeline, list `step_order` explicitly—the GUI always writes it for you.

3.2 Trim signal

GUI panel: Trim signal (checkable, per-step Save)

Crops the recording to a region of interest.

Algorithm.

1. Read `trim_start` (default `0`, the start of the recording) and `trim_end` (default unset, the end of the recording).
2. Clamp boundaries to `[0, signal_duration]`.
3. Call `signal.crop(tmin=trim_start, tmax=trim_end)` if the resulting window differs from the full recording.

Parameter	Default	Description
`trim_start`	`0.0`	Start time in seconds. 0 = beginning.
`trim_end`	`""`	End time in seconds. The TOML default is the empty string (unset); leaving it empty means "end of file".

Memory-preserving (lazy) cropping. `signal.copy().crop(...)` works on MNE's file-backed `Raw` object—`.copy()` is a metadata-only shallow copy and `.crop()` just adjusts the internal sample-range markers.

The full recording is never materialised in RAM. Only the kept sub-range is read from disk on demand, when a downstream step that actually needs data (filtering, ICA fit, etc.) consumes it. The same lazy property holds for `set_montage`, `set_eeg_reference`, and `set_annotations`, which are all pure metadata. As a result, recordings substantially larger than available RAM can be trimmed and processed without ever loading the original full file.

Save default: `[trim] save = false`. For Trim the 🗄 drawer is an **ordinary user toggle** like every other: tick it if you want the cropped signal kept as `<base>-trimmed-raw.fif`, and a config's value is honoured verbatim. Leaving it off costs nothing—a Svarog launch or an eye click that needs the cropped file writes it to that same canonical path on demand. When present, that snapshot is the canonical cropped signal reused for Svarog launches from the Preprocessing tab (see below).

Eye button 👁. Opens the **Preview & Trim** window directly (full-recording overview with the trim region shaded; click on the canvas to set start/end, or type values; `+/-` amplitude controls and a DC-removal checkbox). It does

not open a Svarog split view because Svarog's split-sync aligns by sample index, which left the cropped FIF's time axis labelled `00:00` and the original at `trim_start`—visually misleading when the alignment was actually correct.

Trim is the source of truth for "what the pipeline sees." Every "open in Svarog" button on the Preprocessing tab—the step 👁 eyes and the 🔒/✏ decision buttons that open a bad-channels / ICA / segments review on demand—feeds Svarog the **cropped segment**, not the full input. When a fresh `<base>-trimmed-raw.fif` snapshot (matching `[cfg:<hash>]`) is already on disk it is reused; otherwise the cropped segment is written to that canonical `saved_steps/<base>-trimmed-raw.fif` path in the project output tree (never system-temp) and left in place for later launches to reuse.

3.3 Detect bad channels

GUI panel: Detect bad channels (checkable, per-step Save)

Automatically identifies noisy, flat, or uncorrelated EEG channels that would degrade later processing (filtering, ICA, epoch averaging). Detected channels are then **dropped from the signal** via `signal.pick(picks=None, exclude="bads")` in `_step_bad_channels`—they are removed, not interpolated from neighbours.

Algorithm—five robust detectors + iterate-once cleanup. IDE4EEG implements five parallel detectors—flat (SD-based), flatline duration, robust deviation, windowed correlation, HF noise—plus a one-pass cleanup that re-runs the cross-channel detectors after the obvious bads are dropped. The five criteria are toggleable independently; four ship enabled by default (flat, flatline duration, robust deviation, HF noise) while **windowed correlation is off by default** (opt-in—on standard 19-ch 10-20 montages the peripheral ring naturally carries $r \approx 0.7$-0.9, so it over-flags). They run sequentially on a single shared pre-conditioning of the signal (notch + 1 Hz high-pass) so the thresholds are comparable across recordings. Algorithm and default values follow the PREP pipeline (Bigdely-Shamlo et al. 2015, doi:10.3389/fninf.2015.00016) for criteria 1, 3, 4, 5; Criterion 2 (flatline duration) mirrors EEGLAB `clean_flatlines.m` (Mullen et al. 2015). See § What IDE4EEG implements vs. PREP below for the precise correspondence and docs/literature-reviews/EEG_bad_channels.pdf for the cross-package + literature survey.

Per-channel rejection reasons. Each flagged channel keeps the **criterion/criteria that flagged it** (a channel can trip several—e.g. a flat channel also de-correlates). These are: (1) printed in the Run log—`Bad channels final list (2 channels): O2 (correlation, hf_noise); Fp1 (deviation)`; and (2) written into the Svarog preselect `.tag` as a `desc.reason` field per channel (display only—never the channel `name`, so the editable review's name round-trip is unaffected), so the bad-channel review can show *why* each channel was flagged. Channels added by the iterate-to-convergence loop are attributed to the criterion that caught them. (Whether the installed Svarog jar renders `desc.reason` depends on its version—the IDE4EEG side writes it; a small Svarog render is the remaining piece.)

Preconditioning (shared by all five detectors)

1. **Notch** at the line frequency (50 Hz default for Europe / most of Asia / Africa / Australia; 60 Hz for the Americas / Japan; inherited from `filters.notch_freq` when configured). With **harmonics** enabled (default ON), the notch removes the fundamental AND every integer harmonic up to Nyquist (50, 100, 150, … or 60, 120, 180, …).
2. **High-pass** at `hp_freq_hz` (default 1.0 Hz). DC drift dominates raw amplitude and destroys correlation; the 1 Hz HP is load-bearing for the cross-channel statistics.

   Robust statistics—what MAD means. Every detector below uses the **median absolute deviation** instead of the ordinary standard deviation: $\mathrm{MAD}(x) = \mathrm{median}_i(|x_i - \mathrm{median}(x)|)$—the median distance of the samples from their median. Scaled by $1.4826 = 1/\Phi^{-1}(0.75)$, it becomes a consistent estimator of the Gaussian σ:

$$\sigma_{robust} = 1.4826 \cdot \mathrm{MAD}$$

Detector 1—Absolute amplitude checks

> **If** median SD for all channels $\in$ [`recording_amp_check_min_uV`, `recording_amp_check_max_uV`] µV then check for each channel SD $<$ `flat_sd_threshold_uV` µV.

The sanity check is the precondition for the per-channel SD floor. The median across channels of the robust SD ($1.4826 \cdot \mathrm{MAD}$) is compared against the configured envelope (default **[0.5, 200] µV**). Out-of-range strongly suggests a calibration error—a wrong `calibrationGain` factor, units recorded in `V` instead of `µV`, or a saturated

amp—rather than every channel being simultaneously dead. IDE4EEG logs a `WARNING` with the actual median and the envelope, and the per-channel SD check **is skipped for that run only**.

The other criteria (Flatline / Amplitude outlier / Correlation / HF noise) are calibration-insensitive and keep running.

When the median is in range, each channel's robust SD is compared against `flat_sd_threshold_uV` (default **1e-3 µV** = 10^{-9} V, matching PREP MATLAB's `findNoisyChannels.m:289`). Channels below the floor are flagged. Catches numerically-flat / silent-ADC channels.

> **The two checks share one master toggle in the GUI** because neither makes sense in isolation: the per-channel SD floor is brittle against calibration errors, and the median-SD sanity check exists specifically to gate it.

Detector 2—Flatline duration (eps-relative)

Mirrors EEGLAB `clean_flatlines.m` (Mullen et al. 2015): for each channel, find the longest contiguous run of samples where `|x[i+1] − x[i]| < max_jitter × machine_precision` (i.e. samples are essentially identical at the data's numerical precision). A channel is flagged if its longest such run meets or exceeds `flatline_max_duration_s` (default 5 s; `max_jitter` default 20).

> **What "machine precision" means.** `machine_precision = np.finfo(dtype).eps`—the smallest distinguishable difference in the data's number format. On float64 (MNE default) it's ~2.2×10^{-16} V, so at `max_jitter = 20` the test asks *"do these two samples differ by less than ~4.4×10^{-15} V?"*. At that scale this is essentially *"are they bitwise identical?"*—true only for silent ADCs, railed channels, and post-rereference collisions, not for any real EEG. The threshold scales with the data's dtype precision rather than its µV magnitude, so the detector is **calibration-insensitive**: a recording with a wrong `calibrationGain` factor (the wakeEEG.raw case) doesn't trip it. **Caveat:** float32 data shrinks the safety margin to ~2.4 µV; rare in MNE workflows but worth knowing if your file reader produces float32.

Complements Detector 1: same defect class (channel is flat) but a fundamentally different threshold family (duration vs. amplitude). Catches silent ADC, railed ADC, post-rereference collision, and any literally-constant stretches.

Detector 3—Amplitude outlier (PREP: "Robust deviation")

For every EEG channel, compute the robust SD. Then compute a **robust z-score across channels** using the median + MAD of the per-channel SDs:

$$z(c) = \frac{\sigma_{robust}(c) - \text{median}_c(\sigma_{robust})}{1.4826 \cdot \text{median}_c(\,|\sigma_{robust} - \text{median}_c(\sigma_{robust})|\,)}$$

Flag any channel with `|z(c)| > deviation_z_threshold` (default 5.0). Catches gross under- and over-amplitude channels. The MAD-based cross-channel statistic ensures a single bad channel cannot poison the reference used to judge the others.

Detector 4—Windowed correlation (max over other channels)

Split the recording into `correlation_window_seconds`-long (default 1.0 s) non-overlapping windows. For every window `w` and every channel `c`, compute the maximum absolute Pearson correlation between `c` and any other EEG channel:

$$\rho(c, w) = \max_{c'} \; |\,\text{corr}(x_c, x_{c'})\,|_w \qquad (c' \neq c)$$

A window is "bad" for `c` when `ρ(c, w) < correlation_threshold` (default 0.3, lowered from PREP's 0.4—see docs/literature-reviews/EEG_bad_channels.pdf). Channel `c` is flagged when the fraction of bad windows exceeds `bad_window_fraction_threshold` (default 0.05, i.e. 5 %). The fraction-of-bad-windows form—instead of a single full-record correlation—means a transient artifact does not doom a channel for the whole record.

> **Not montage-aware.** Despite the intuition that volume-conducted EEG channels should correlate most strongly with their spatial neighbours, this criterion uses every other EEG channel as a candidate, not a k-nearest-neighbour subset. The best-correlated channel will *physically* usually be a spatial neighbour, but the algorithm doesn't enforce it. The original PREP MATLAB has a kNN-with-electrode-positions variant; pyprep and IDE4EEG both drop the kNN to avoid the montage requirement.

Two safety bailouts. The criterion is **skipped** when there are fewer than 3 EEG channels—the `max correlation against OTHERS` statistic becomes degenerate (with exactly 2 channels, both share the same value and pass-or-fail together). It also **bails out and flags nothing** when *every* channel exceeds the bad-window fraction—that's a sign the detector has degenerately failed (no good reference left). Both bailouts log a WARNING explaining the cause and suggesting remediation (lower `correlation_threshold`, disable the criterion, or investigate the recording). The other three detectors still run and contribute their findings.

Detector 5—High-frequency noise ratio

Compute the robust SD on the HF-filtered signal (above `hf_lower_hz`, default 50 Hz) and on the broadband signal; take their ratio per channel:

$$\eta(c) = \frac{\sigma_{\text{robust\_HF}}(c)}{\sigma_{\text{robust\_broadband}}(c)}$$

Then a robust z-score across channels of $\eta(c)$, and flag any channel whose $z >$ `hf_noise_z_threshold` (default 5.0)—only positive deviations (high HF is bad; low HF is normal). Catches bridged electrodes and EMG-contaminated channels that may pass the amplitude and correlation checks. The amplitude-normalised form (a *ratio*, not absolute HF power) is the reason this is orthogonal to the deviation detector: a loud-but-clean channel passes; a quiet-but-EMG-contaminated channel doesn't.

Refinement passes—"Iterate up to N times" (default 1)

After the first-pass union of flagged channels is taken, drop them and re-run the three robust cross-channel detectors (deviation, correlation, HF) on the cleaner reference, repeating until a pass finds nothing new or the cap is hit. This catches successive layers of borderline channels masked by the obvious outliers—the cross-channel MAD shrinks once the worst offenders are out, surfacing channels that were just under the $z = 5$ threshold the previous time around (the original PREP paper demonstrates this iteration is the single biggest robustness gain in their pipeline; PREP itself iterates to convergence, capped at 4). The "Iterate up to [N] times" pulldown sets the cap via `max_iterations` (0–4): `0` = single pass (no refinement), `1` = the recommended default, `4` = PREP's cap. The loop only ever *adds* channels, so it always terminates and stops early on convergence.

Cross-coverage by design

The above five criteria are complementary, not orthogonal: a clearly-flat channel (literal zeros) also fails the correlation detector (zero-variance ⇒ zero correlation with anyone), and a severely over-amplitude channel may trip both deviation and HF noise.

Manual review (on-demand)

This step has no 👁, because marking a channel changes no samples, so the decision is the only thing to show. To inspect or change the auto-detected list, use the **decision button** on the Detect bad channels panel header: 🔒 opens the picks read-only, ✏️ opens them editable, if **Allow manual modifications** on the Config tab is set. Clicking the button first **runs the pipeline up to this step**, with the detector forced to re-run, so the picks you are shown reflect your *current* settings rather than a cached snapshot—expect a wait and a burst of Run-log output before the window appears. Then Svarog opens with the signal + the auto-detected channels pre-marked; tick/untick, close, and your selection is **remembered for this recording in the current session** (absolute-replace—it fully replaces the auto list, and un-marks correctly drop channels). A ✋ **marker** appears on the panel; every later Run of the same file re-applies your selection until you revert it (click ✋ → **Revert to automatic**). It is **not** written to the config. To walk every editable step in one run, arm **stop at each user-editable step** (Config tab).

Top-level parameters

Parameter	Default	Description
`choose_bad_channels`	*(absent)*	**Legacy / ignored.** The step detects whenever it runs, so there is no mode to choose. The key is **popped at config load** with a warning (`ide4eeg/config_schema.py:migrate_legacy_keys`) and never applied; present only in old configs. To skip detection, untick the **Detect bad channels** panel header (equivalently, drop `bad_channels` from `[preprocessing] step_order`)—that is what `"none"` meant. To drop a specific set of channels outright, list them in `[choosing_channels] dropped_channels` on the Input tab; because this step ends in `pick(exclude="bads")` it *removes* the channels it flags, so an explicit list is the same operation.
`manual_bad_channels`	*(absent)*	**Legacy / ignored.** Manual bad-channel decisions now live in the GUI session (RAM), not the config; this key is **stripped at config load** and never applied. Present only in old configs. Not a GUI-typed field.

Output. When `save = true`, the bad-channels step writes the cleaned signal as `<base>-bad-marked-raw.fif`. The config is updated with the flagged channel list; the signal's `info["bads"]` is set accordingly; then `signal.pick(picks=None, exclude="bads")` drops the channels from the working signal. (Diagnostic per-criterion plots are a planned follow-up—the legacy detector wrote `bad_channels_detection/*.png` per-channel band-power plots; the current detector does not yet write equivalents. The Run-log diagnostic table—one line per channel per criterion with the determining value and `[FLAGGED]` marker—partially fills the same role.)

What IDE4EEG implements vs. PREP

IDE4EEG is **substantially PREP-aligned but not literally complete**. Four of the five detectors use PREP's published thresholds exactly (criterion 2, Flatline duration, is from EEGLAB `clean_flatlines.m`, not PREP). Three further differences from the canonical PREP algorithm:

Aspect	PREP (Bigdely-Shamlo 2015)	IDE4EEG
Flat detection	robust SD below floor	**same** (`_detect_flat_channels`)
Robust deviation z-score	`abs(z) > 5` across channels	**same** (`_detect_deviation_channels`)
Windowed correlation	1-s windows, `abs(r) < 0.3`, >5 % bad	**same** (`_detect_correlation_channels`)
HF/broadband SD ratio	`z > 5` above 50 Hz	**same** (`_detect_hf_noise_channels`)
Pre-conditioning HP	1 Hz	**same** (`prep.hp_freq_hz`)
Iterate-once cleanup	iterate **until convergence**	**one extra pass only**—captures most of the benefit without the convergence-edge-case logic
RANSAC interpolation criterion	included; strongest single criterion	**not implemented**
Derived SNR criterion (corr × HF)	included	**not implemented**—redundant given we have both components individually
Robust-average reference during assess.	rereferences inside the loop	**not done**—Reference is a separate user-controlled reorderable step

Advanced parameters (`[choosing_channels.prep]` sub-block): see Appendix E.2.

3.4 Reference

GUI panel: Reference (checkable, reorderable, per-step Save). **Backend:** `_set_reference` (`channels_and_signal.py`)

Re-references the signal—i.e. subtracts a reference signal from every channel—using `signal.set_eeg_reference(...)`. Independent of the montage (set on the Input tab): re-referencing alters signal values, not electrode positions. The montage is already applied before this step runs.

Six options:

Preset	Formula	Description
Raw signal (no re-referencing)	`x_i(t)` unchanged	Keep the recording's native reference. **Default.**
Common Average (CAR)	`x_i'(t) = x_i(t) − (1/N) · Σ x_j(t)`	Subtract the mean across all good EEG channels.
Linked mastoids	`x_i'(t) = x_i(t) − mean(M1, M2)`	Average of mastoid channels.
Linked ears	`x_i'(t) = x_i(t) − mean(A1, A2)`	Average of ear channels.
REST (infinity)	Yao (2001) standardisation	Reconstruct a reference-free potential via the lead-field matrix.
Custom	`x_i'(t) = x_i(t) − mean(ref_channels)`	Comma-separated list of channels—their mean becomes the reference, and they are added to `info["bads"]` so subsequent steps and channel filters ignore them.

Parameter	Default	Description
`re_reference`	`[]`	`[]` / `""` / `"None"` (raw—all three accepted), `"average"` / `"CAR"` (common average), `"REST"` (infinity), or a list / comma-string of channel names (custom).

When `save = true`, the rereferenced signal is written to `<base>-rereferenced-raw.fif`. The 👁 button shows the before/after comparison.

> **Reference:** Yao D (2001) "A method to standardize a reference of scalp EEG recordings to a point at infinity." *Physiol. Meas.* 22:693.

3.5 Resample

GUI panel: Resample (checkable, reorderable, per-step Save)

Downsamples the signal to a lower sampling frequency, reducing data size and computation time without losing information below the target Nyquist frequency. Uses MNE's `signal.resample(sfreq=target)`, which internally applies a low-pass anti-aliasing FIR filter at the new Nyquist frequency, then resamples via MNE's default FFT-based method. A signal already at the target rate is left unchanged; the step resamples whenever the rate differs, so a target *above* the current rate would upsample.

Parameter	Default	Description
`resample_freq`	`0`	Target sampling frequency (Hz). `0` = keep original rate.

3.6 Filters

GUI panel: Filters (checkable, reorderable, per-step Save)

In the default IIR path only EEG channels are filtered (STIM and auxiliary channels are bypassed); the FIR path delegates to MNE, which filters all data channels (EEG plus EOG/EMG/ECG) but leaves STIM untouched.

IIR (Butterworth / Chebyshev II)—default. Filters are designed automatically from the cutoff frequency and the signal's sampling rate using `scipy.signal.iirdesign`. Stored in second-order sections (SOS) format for

numerical stability and applied zero-phase via `sosfiltfilt` (bidirectional, no phase distortion). Note that the forward+backward pass squares the filter's magnitude response—a `-3 dB` design becomes `-6 dB` at cutoff—and doubles the effective filter order. The filter-response plots produced by `plot_filt = true` show the single-pass design, not the realised doubled response.

Filter	Passband edge	Stopband edge	Design
Highpass	`f_hp` Hz	`f_hp / 2` Hz	Butterworth, gstop=10 dB, gpass=3 dB
Lowpass	`f_lp` Hz	`min(2*f_lp, 0.95*Nyquist)` Hz	Butterworth, gstop=12 dB, gpass=3 dB
Notch	`f_notch ± 2.5` Hz	`f_notch ± 0.1` Hz	Chebyshev II, gstop=25 dB

FIR (MNE windowed sinc). Uses MNE's `Raw.filter(..., method="fir")` / `Raw.notch_filter(..., method="fir")` with a Hamming-windowed sinc design and automatic filter length. Always stable, linear phase, zero group delay. Applied via FFT overlap-add. Slower than IIR but has no stability concerns.

SOS vs ba format. SOS decomposes the filter into cascaded biquad stages, numerically stable for high-order filters. The older `ba` (numerator/denominator polynomial) format can produce unstable filters due to floating-point errors, especially at high sampling rates.

Parameter	Default	Description
`plot_filt`	`false`	Save filter response plots to disk.
`show_filt`	*(absent)*	**Legacy / ignored.** Once a checkbox meaning "pop up the frequency response *during the run*"; that was replaced by the on-click **Show filter plots** button below, because a run should not open windows. Present only in old configs, where it is silently ignored (a run never displays a plot). Nothing to migrate—use the button, or `plot_filt` to write the plots to disk.
`method`	`"iir"`	`"iir"` (Butterworth/Chebyshev) or `"fir"` (MNE).
`highpass_freq`	`0.5`	Highpass cutoff Hz. Default `0.5` Hz; `0` = off. Common: 0.1, 0.5, 1.0 Hz.
`lowpass_freq`	`0`	Lowpass cutoff Hz. `0` = off (default). Common: 30, 40, 100 Hz.
`notch_freq`	`50`	Notch centre Hz. Default `50` (Europe/Asia); use `60` (Americas); `0` = off.
`notch_harmonics`	`true`	The **harmonics** checkbox next to the notch field. When ON, the notch removes the fundamental AND every integer harmonic up to Nyquist (50/100/150/… or 60/120/180/…). **On by default**; no effect when `notch_freq = 0`.

Output. When `plot_filt = true`, frequency response plots are saved in `filters_plot/` folder.

Viewing the response. To see the response without running anything, use the filter panel's **Show filter plots** button, which designs and displays it for the current settings; the adjacent **Scale** dropdown switches the magnitude axis between `dB` and `linear`. This is GUI-only and has no config key—a run never opens a plot window.

3.7 MP filter

GUI panel: MP filter—checkable, per-step Save.

Reconstructs the signal from a selected subset of Matching Pursuit atoms, implementing a nonlinear filter. Unlike conventional linear filters which operate in the frequency domain, MP filtering can select atoms based on combined time-frequency-energy criteria.

MP filter runs its own internal decomposition, using segments in rest mode, regardless of the pipeline's segmentation mode. It produces a separate decomposition `.db` book at `<output>/mp_filter/book_<base>_filter.db`, while the next step MP Decomposition saves its book at `<output>/mp_decomposition/`. The two coexist; downstream

analyses (EEG Profiles, MP-Dipoles) can choose either via the per-analysis `mp_book_source` setting on their Analysis-tab panels; The book pulldown on those panels offers also an option **"Use existing book file..."**, which opens a file picker so you can point the analysis at any pre-existing `.db` book on disk (e.g. one produced in an earlier session, or by hand) without re-running a decomposition step. This sets `mp_book_source = "file"` and stores the chosen path in `mp_book_file`. For MMP-dipoles the picked book must be an MMP1 book—this is verified when the book is opened (from its stored algorithm metadata).

Algorithm.

1. Resolve the internal-decomp window length: `[mp_filter].window_length` (explicit override) → `[rest].window_length` (inherited) → a whole-signal window when `[rest].whole_signal` is set → `20 s` (final fallback).
2. Cut the signal into non-overlapping `window_length`-second rest windows from sample 0.
3. Run empi on the concatenated buffer using `[mp_filter.decomp]` parameters (same shape as `[matching_pursuit]`: algorithm, iterations, explained_energy, channels, etc.). Skip empi when the on-disk book's `_input_hash` matches the freshly-computed `_compute_mp_filter_book_hash` digest (reuse semantics analogous to standalone MP—editing reconstruction-only `[mp_filter.filter]` fields never invalidates the book).
4. For each window and channel, filter the atom list by user-specified `[mp_filter.filter]` criteria (frequency range, scale range, amplitude range, time range, energy threshold, iteration cutoff).
5. Depending on `[mp_filter.filter].mode`:
   - `"keep"` (default): reconstruct from matching atoms only—*denoising / band-isolation.*
   - `"remove"`: reconstruct from non-matching atoms (subtract artefact atoms).
6. Replace each window's samples in the Raw object with the reconstruction.

Parameters—internal decomposition (`[mp_filter.decomp]`, mirrors `[matching_pursuit]`)

Parameter	Default	Description
`algorithm`	`"smp"`	SMP / MMP1 / MMP3—see §3.8 table.
`iterations`	`30`	Max atoms per channel/segment.
`explained_energy`	`0.99`	Stop fraction, `0`–`1` **in TOML** (the GUI field shows it as a percent). `1.0` = iteration cap only.
`channels`	`""`	Subset of channels to decompose; empty = all EEG channels (falls back to all non-STIM data channels only if no EEG channels are present). Mirrors §3.8.
`cpu_workers`	`0`	Inherit from `parallelism.n_jobs` (override per §3.8).

Parameters—window length (`[mp_filter]`)

Parameter	Default	Description
`window_length`	`0`	Seconds per internal rest window. `0` = inherit from `[rest].window_length` (default 20 s).

Parameters—reconstruction criteria (`[mp_filter.filter]`, "osc. EEG" defaults)

The preset combo at the top of the criteria table offers **osc. EEG (f>0.5 Hz, w>0.1 s, f·s>0.1)** as the default selection—oscillatory atoms in the physiological EEG band, rejecting slow drift (< 0.5 Hz), impulse-like spikes (< 0.1 s), the 50 Hz line and harmonics (> 49 Hz), and non-oscillatory bumps (< 0.1 oscillation cycles). A second preset, **de-drift (0.1 cyc)**, applies only the cycle floor (`min_cycles = 0.1`)—dropping DC/slow-drift bumps while keeping everything with real periodicity, without the freq/scale bounds. Picking a different EEG-structure preset (Sleep spindles, Alpha, …) populates the same row.

Two schemas—the GUI writes `ranges`. The criteria table accepts several rows (one per atom family), so `[mp_filter.filter]` holds a `ranges` list, one entry per row, each carrying the range keys below:

```
[mp_filter]
  [mp_filter.filter]
  mode = "keep"
  time_min = 0        # these three stay at the top level
  time_max = 0
  min_energy = 0
  ranges = [
    { freq_min = 0.5, freq_max = 49.0, scale_min = 0.1, scale_max = 0.0,
      amp_min = 0.0, amp_max = 0.0, max_iterations = 0,
      min_cycles = 0.0, max_cycles = 0.0 },
  ]
```

A hand-written config may instead put a single range's keys at the top level (the legacy flat form) and omit `ranges`; that is still read, and behaves as one range. Do not mix the two—saving from the GUI drops any flat range keys. `mode`, `time_min`, `time_max` and `min_energy` are per-filter, not per-range, and always live at the top level. An all-zero range matches every atom, i.e. "no constraint"; if every range is all-zero and no time/energy bound is set, the step logs that no criteria are set and leaves the signal unchanged.

Parameter	Default	Description
`mode`	`"keep"`	`"keep"` or `"remove"` matching atoms.
`freq_min`	`0.5`	Atoms with frequency ≥ this (Hz). 0 = no constraint.
`freq_max`	`49`	Atoms with frequency ≤ this (Hz). 0 = no constraint.
`amp_min`	`0`	Atom peak-to-peak amplitude lower bound (µV).
`amp_max`	`0`	Upper bound (µV).
`scale_min`	`0.1`	Atom scale (width) lower bound (s).
`scale_max`	`0`	Upper bound (s).
`time_min`	`0`	Atom-centre time lower bound (s, in-window).
`time_max`	`0`	Upper bound.
`min_energy`	`0`	Atom-energy lower bound.
`max_iterations`	`0`	Only first N atoms (by iteration). 0 = all.
`min_cycles`	`0`	Min oscillation cycles `f·s` (= freq × `scale_s` ≈ cycles across the FWHM). `< 1` = bump/transient, `≥ 1` = oscillation. 0 = no constraint.
`max_cycles`	`0`	Upper bound on `f·s`. 0 = no constraint.

The `min_cycles`/`max_cycles` gate is a **cycle-count** filter, orthogonal to freq/scale: `f·s < 0.5` is a monophasic bump (a slow ERP such as the P300 is one, `f·s ≈ 0.5`), `f·s ≈ 0` is DC/drift, and `f·s ≥ 1` is a genuine oscillation. So `min_cycles = 1` keeps rhythms (spindles `f·s ≈ 6`, alpha, beta) and drops transients, while `min_cycles ≈ 0.1` drops only DC/slow-drift bumps—a **nonlinear** drift cut that spares the signal's own low-frequency content, which a linear high-pass cannot (it would attenuate that content too). Delta (spike) atoms carry no `f·s` and are excluded by any cycle bound.

Book reuse. Editing reconstruction criteria (the `[mp_filter.filter]` sub-block) does not trigger an empi rerun—the filter operates on already-decomposed atoms. Editing the internal decomp parameters or `window_length` DOES invalidate, and the next run re-executes empi. See MP book reuse for the staleness machinery.

3.8 MP decomposition (matching pursuit)

GUI panel: MP Decomposition—checkable, with no 🗄 drawer: the `.db` book is the step's only output and is written whenever the step runs.

Decomposes the EEG signal into a sum of Gabor atoms—Gaussian-windowed sinusoids that are optimally matched to the signal's time-frequency content. The result is an adaptive, parametric representation stored in a persistent SQLite book file (`.db`). This step does not modify the signal. Requires the `empi` binary, which is installed beside the Svarog jar in `~/.obci/svarog/` and auto-detected there (see External tool paths).

Mathematical foundation. Matching Pursuit (Mallat & Zhang 1993) is a greedy iterative algorithm:

1. Start with the full signal as the zero residual: `R_0(t) = x(t)`.
2. At iteration `n`, find the Gabor atom `g_n` that maximises the inner product with the residual: `g_n = argmax_g |<R_n, g>|`.
3. Subtract the weighted atom: `R_{n+1}(t) = R_n(t) - <R_n, g_n> * g_n(t)`.
4. Repeat until residual energy drops below `(1 - explained_energy)` of the original, or `iterations` atoms are extracted.

Each Gabor atom is parameterised as:

```
g(t) = A * exp(-pi * (t - t0)^2 / s^2) * cos(2*pi*f*t + phi)
```

where `A` = amplitude, `t0` = centre time, `s` = scale (temporal width), `f` = frequency, `phi` = phase.

> **What `s` (scale) means—it is ≈ FWHM, not the standard deviation.** Because of the `pi` in the exponent, `s` is **not** the Gaussian σ. Relative to a textbook `exp(-x^2 / (2*sigma^2))`: `sigma = s / sqrt(2*pi) ≈ 0.40*s`, and the full width at half maximum `FWHM = 2*sqrt(ln2/pi)*s ≈ 0.94*s`. So **`s` is very nearly the FWHM—the visible atom duration**, and that is exactly the quantity the book's `scale_s` and the atom-filter `scale_min` / `scale_max` fields use. Set those fields as **durations, not σ**: e.g. `scale_max = 0.5` keeps atoms up to FWHM ≈ 0.47 s (σ ≈ 0.20 s), so to keep a ~0.5–0.9 s ERP component (e.g. the P300) use `scale_max ≈ 1.0`, not `0.5`.

Multivariate MP (MMP) extends the algorithm to multiple channels simultaneously: at each iteration, the atom parameters (t0, s, f) are shared across all channels, while amplitude and phase are fitted per channel. This produces "macroatoms" that describe coherent multi-channel activity.

empi (Różański 2024) is a GPU-accelerated implementation that uses continuous parameter optimisation.

Algorithm types.

Algorithm	Key	Description
SMP	`smp`	Single-channel MP: each channel decomposed independently.
MMP1	`mmp1`	Multivariate MP, constant phase across derivations.
MMP3	`mmp3`	Multivariate MP, variable phase.

> **MMP requires ≥2 channels.** MMP1 and MMP3 are *multivariate*—they fit atoms shared across channels, so they need at least two channels to decompose. On single-channel input (a single derivation, or `matching_pursuit.channels` narrowed to one) `empi` rejects an MMP run and exits with an error. IDE4EEG's preflight catches this before the run and asks you to switch to **SMP** or select more channels—SMP (each channel decomposed independently) is the correct choice for single-channel recordings.

Parameters exposed in the GUI (Preprocess tab → MP Decomposition panel):

Parameter	TOML key	Default	GUI control	Description
Algorithm	`matching_pursuit.algorithm`	`"smp"`	dropdown	SMP / MMP1 / MMP3—see the table above.
Channels	`matching_pursuit.channels`	all EEG	channel-pick panel	Subset of channels to decompose; empty = every EEG channel.
Iterations	`matching_pursuit.iterations`	`80`	spin	Maximum atoms to extract. `0` = no cap (then `explained_energy` is the only stop condition).

Parameter	TOML key	Default	GUI control	Description
Explained energy	`matching_pursuit.explained_energy`	`1.0`	percent edit	Stop when this fraction of total energy is explained. **TOML stores a `0–1` fraction; the GUI field is a percent** (`1.0` ↔ 100 %). `1.0` = stop only at the iteration cap.
CPU workers	`matching_pursuit.cpu_workers`	inherit	spin (`0` = inherit)	Override for empi worker count—see the **CPU workers** note below.

Fixed defaults (round-tripped through `config.toml` but not editable from the GUI):

TOML key	Value	Why fixed
`matching_pursuit.optimization`	`"global"`	Continuous-dictionary refinement gives the best atom localisation accuracy; the `"local"` and `"none"` paths exist in empi for benchmarking against legacy implementations. Change in the TOML only if you know why.
`matching_pursuit.energy_error`	`0.01`	Dictionary density (ε^2). Smaller = denser dictionary = more precise but quadratically slower. `0.01` is tuned for the accuracy / runtime balance of typical EEG.

CPU workers: empi has its own override, separate from `parallelism.n_jobs`.

Empi's parallelism uses a fundamentally different cost model than the joblib/loky workers that drive every other parallelisable step (MNE filter / resample, TFR, …). The pipeline-level `parallelism.n_jobs` heuristic (see Parallel jobs) caps worker count by `ram_gb // 2` because each loky worker spawns a fresh Python interpreter—typically 200–500 MB of resident memory per worker once MNE + NumPy are loaded.

Empi runs as a single C++ subprocess; its workers share the atom dictionary and signal buffer through one address space. Total RAM is roughly 100 MB regardless of how many workers you configure—there is no per-worker Python interpreter to pay for. So a machine that's appropriately set to `parallelism.n_jobs = 4` for the rest of the pipeline can comfortably run `matching_pursuit.cpu_workers = 8` (or as many as the CPU has physical cores).

Resolution order at empi invocation:

1. `matching_pursuit.cpu_workers > 0`—explicit user override (takes precedence).
2. Otherwise, `parallelism.n_jobs` from `[parallelism]` (the unified pipeline knob).
3. `1` as the final fallback for tests and callers that didn't inject anything.

Each empi worker runs single-threaded (`--cpu-threads 1`); Gabor-dictionary matching is not BLAS-heavy, so adding BLAS threads per worker hurts more than it helps.

Output. SQLite `.db` book file containing all atoms with parameters: segment, channel, iteration, frequency, scale, amplitude, energy, phase. Consumed by EEG profiles and dipole fitting when those analyses have `mp_book_source = "decomposition"` (the default). MP filter has its OWN internal book produced independently—see §3.7.

MP book reuse—hash-controlled. MP decomposition is one of the most expensive preprocessing steps (minutes to hours). To avoid recomputing when nothing relevant has changed, MP follows a hash-controlled reuse policy. The freshness check runs whether or not `mp_decomposition` is in `step_order`. See MP book reuse for the full policy table.

References:

Mallat SG, Zhang Z (1993) "Matching Pursuits with Time-Frequency Dictionaries." *IEEE Trans Signal Process* 41(12):3397–3415. DOI: 10.1109/78.258082

Kuś R, Różański PT, Durka PJ (2013) "Multivariate matching pursuit in optimal Gabor dictionaries." *BioMed Eng OnLine* 12:94. DOI: 10.1186/1475-925X-12-94

Różański PT (2024) "empi: GPU-Accelerated Matching Pursuit with Continuous Dictionaries." *ACM Trans Math Softw* 50(3):17. DOI: 10.1145/3674832

3.9 ICA (independent component analysis)

GUI panel: ICA (checkable, per-step Save, View Step Result)

Backend: `ide4eeg/preprocessing/ica.py::fit_and_apply_ica`

Independent Component Analysis separates the EEG into statistically independent sources. Components flagged by the selector as artifacts (eye blinks, eye movements, muscle, cardiac, line noise, channel noise) are projected out, preserving the brain activity in the remaining subspace.

The TOML section is named `[ICA_EOG]`. Despite the name, the default selector is ICLabel **when an ICLabel backend (onnxruntime or PyTorch) is installed**—otherwise it falls back to the backend-free `find_bads` heuristic. ICLabel flags eye, muscle, heart, line noise, channel noise, and "other" in addition to EOG.

Precision note on attribution. The bullets and table below are deliberately specific about who implements what. `mne.preprocessing.ICA` is a dispatcher and state container; the numerical kernels for the three `method=` choices live in three different places (one in MNE in-tree, two in separately-released upstream packages—`python-picard` and `scikit-learn`'s `FastICA`), and the ICLabel selector lives in yet another separate package (`mne_icalabel`) that the MNE project maintains but does not ship as core. IDE4EEG's `ica.py` is a workflow wrapper around all of these—it ships no original numerical ICA code. We try to over-credit nothing.

Algorithm.

1. **Fit-time filter copy.** A `raw.copy()` is taken and filtered with three independent knobs (all applied to the COPY only; the original signal—and therefore the cleaned signal returned by `ica.apply`—never sees these filters). **Code source:** `raw.copy().filter(...)` / `raw.copy().notch_filter(...)` in MNE-Python in-tree (Gramfort 2014). **Methodological source for the 1 Hz default:** Winkler 2015 (high-pass filtering an ICA fit copy near 1 Hz substantially improves ocular-component identification without losing slow ERPs in the analysis signal).
   - `fit_highpass_hz` (default 1.0 Hz, Winkler 2015's classical recommendation; set 0 to disable).
   - `fit_lowpass_hz` (default 0 = off; set to drop muscle / line-noise harmonics from the ICA fit while keeping a wider analysis band).
   - `fit_notch_hz` (default 0 = off; set to the local line frequency (50 / 60 Hz) to knock out line noise from the fit copy if the analysis filter chain doesn't notch). Whether the fundamental's harmonics are also removed is the `fit_notch_harmonics` boolean (default `true`); when ON, the notch is applied at the fundamental AND every integer harmonic up to Nyquist (50/100/150/… or 60/120/180/…). Mirrors the bad-channels preconditioning `notch_harmonics` convention.
2. **Pre-fit reject gate.** Segments whose peak-to-peak amplitude exceeds `reject_uv` (default 500 µV) are dropped from the fit via `ica.fit(..., reject=dict(eeg=...), tstep=reject_tstep)`, so rare big transients (electrode pops, subject coughs) don't dominate the decomposition. `reject_tstep` (default 2.0 s) is the length of the sliding window MNE scores each segment against; shorter windows reject more aggressively. **Code source:** `mne.preprocessing.ICA.fit` (MNE-Python in-tree, Gramfort 2014). **Methodological background:** Delorme & Makeig 2004 (EEGLAB) and Onton et al. 2006—peak-to-peak rejection of high-amplitude transients before ICA is the community-standard pre-ICA hygiene step. IDE4EEG-side contribution: the `reject_uv` scalar µV ergonomic shim that fans out to MNE's `reject=dict(eeg=...)` dict in V.
3. **Pre-whitening (PCA).** Before the chosen decomposition algorithm runs, `mne.preprocessing.ICA.fit` internally:
   1. picks channels (eeg / mag / grad as configured),
   2. applies the `reject=dict(eeg=...)` peak-to-peak gate above to build the training segments,

3. applies `decim` if > 1,
4. mean-centres,
5. runs a PCA decomposition and whitens by the principal components scaled to unit variance, projecting to the chosen number of components (the user's int, the float-fraction-of-variance, or `mne.compute_rank()`'s rank estimate when `n_components="rank"`).

The whitened, dimension-reduced data is what gets handed to Picard / Infomax / FastICA in step 4. ICA's mathematical contract requires whitened input (zero mean, unit variance, decorrelated)—without pre-whitening, none of the three algorithms converge correctly. **Code source:** the PCA whitening and dimension-reduction step inside `mne.preprocessing.ICA.fit` (MNE-Python in-tree, Gramfort 2014), with the rank estimator `mne.compute_rank()` in the same package. **Methodological source for the whitening requirement:** Hyvärinen & Oja 2000—the canonical exposition of ICA's algorithmic family, including why whitening must come before the decorrelation-→-independence step.

PCA whitening with the right `n_components` also doubles as the **rank-defense step**: average-reference EEG has rank `n - 1` (the average constraint removes one degree of freedom), so feeding 32 components into ICA on average-referenced 32-channel data would build a singular mixing matrix. The IDE4EEG default `n_components="rank"` calls `mne.compute_rank(fit_copy)` (SVD-based) to detect this automatically and pass the correct value. Integer and float-fraction overrides are also accepted.

4. **Decomposition.** The whitened data is then handed to one of three numerical kernels, selected by `method`. The default is `"picard"` with `fit_params=dict(ortho=False, extended=True)`, which optimises the same extended-Infomax objective as Lee et al. 1999 but 3–10× faster via preconditioning.

   - `method="picard"` → **upstream `python-picard` package** (separate PyPI release; MNE only dispatches into it). Algorithm: Ablin, Cardoso & Gramfort 2018.
   - `method="fastica"` → `sklearn.decomposition.FastICA` (MNE delegates into scikit-learn). Algorithm: Hyvärinen 1999.
   - `method="infomax"` → **MNE-Python in-tree** at `mne/preprocessing/infomax_.py`. Algorithm: Bell & Sejnowski 1995; with `fit_params(extended=True)` the extended variant of Lee, Girolami & Sejnowski 1999 (the objective Picard accelerates).

   IDE4EEG-side contribution to this step: BLAS threads are temporarily uncapped via `threadpoolctl.threadpool_limits(limits=None)` for the duration of the fit so Picard's inner loop (and the BLAS calls inside Infomax / FastICA) can use all cores, regardless of any earlier thread cap. After the fit, the previous limit is restored.

5. **Auto-detection** (`selector`). Once the decomposition is fitted, components are scored against one of two automatic labelling routes (or their union):

   GUI presentation. The selector is two checkable groupboxes—**ICLabel** (`iclabel`) and **MNE detectors** (`find_bads`); ticking both yields the union (`both`), noted inline (greyed) in the "Components to remove." header above them (that header states the remove polarity once, so neither box title repeats it; its tooltip spells out the union). The ICLabel box holds the seven class ticks, the MNE-detectors box the three detector ticks (muscle / ECG / EOG). **ECG** greys out when the signal has no ECG/MEG channel. **EOG** is signal-adaptive: with a real EOG channel the tick uses it directly; with none, the tick greys and a "choose proxy EOG:" multi-select appears, listing the frontal EEG channels **present in the loaded file** (Fp*/AF*/F*)—picking ≥1 enables EOG-via-proxy (it is the on/off control in that case). No channel names are assumed a-priori; the picker is empty until you choose. TOML configs keep the canonical lowercase `selector` values used throughout this section.

   Both boxes mark what to *remove*—and `both` is a union, not an intersection. Everything you tick in either box is projected out of the signal: ICLabel marks whole classes, the MNE detectors flag specific artifact types. `selector="both"` takes the **union** of the two removal sets (MNE's accumulate-into-`ica.exclude` idiom)—a component is dropped if *either* box marks it. Union (∪ = "or" / sum), **not** intersection (∩ = "and" / overlap): ticking both boxes removes **more**, never less. This is the one thing worth double-checking when reading someone else's config.

GUI ticks vs the TOML key—they are complements. The ICLabel box shows the classes to **remove**, but the TOML key `iclabel_keep` stores the classes to **keep**. So the default `iclabel_keep = ["brain", "other"]` appears in the GUI as the *other five* classes ticked (muscle, eye, heart, line_noise, channel_noise). The GUI inverts on load and on save, so the two never disagree—but if you hand-edit a config, remember you are writing the keep list even though the panel reads "remove". (The key kept its original name deliberately: renaming it would invalidate every saved step snapshot and MP book for a change that alters no signal.)

- `selector="iclabel"`—uses the **upstream `mne_icalabel` package** (separate PyPI release, MNE-affiliated but not part of MNE core). MNE-ICALabel's CNN classifies every component as `brain`, `muscle`, `eye`, `heart`, `line_noise`, `channel_noise`, or `other`. The classifier copy is bandpassed to 1–100 Hz and re-referenced to average first (ICLabel's preconditions). Components whose top-1 class is not in `iclabel_keep` (default `["brain", "other"]`) are removed. **Methodological sources:** Pion-Tonachini, Kreutz-Delgado & Makeig 2019 (the original ICLabel classifier, dataset, and labelling protocol—originally shipped as a MATLAB/EEGLAB plugin); Li, Feitelberg, Saini, Höchenberger & Scheltienne 2022 (the Python port we actually import as `mne_icalabel.label_components`). **Inference backend:** the CNN runs on either **upstream `onnxruntime`** or **upstream `pytorch`**—both optional dependencies, neither in MNE core. When no backend is installed, the GUI disables (greys) the ICLabel groupbox with an orange note and falls back to the MNE detectors, and a pre-launch check (`hard.tools.iclabel_requires_backend`) blocks a config that still requests ICLabel. Install with `pip install onnxruntime` (≈ 30 MB) or `pip install ide4eeg[iclabel]`; PyTorch (≈ 1 GB) also works if it's already on the system for other reasons.
- `selector="find_bads"`—runs three **MNE-Python in-tree** detectors against the reference channels named in `find_bads_eog_ch`:
  - `ICA.find_bads_eog`—Pearson correlation between component time courses and the EOG reference channels, plus adaptive z-scoring. **Methodological source:** community-standard heuristic; MNE's docstring cites no specific paper for this branch.
  - `ICA.find_bads_ecg`—cross-trial phase statistics (default) or Pearson correlation against the ECG channel. **Methodological source:** Dammers et al. 2008 for the phase-statistics route.
  - `ICA.find_bads_muscle`—three-criterion detector (log-log spectral slope + peripheral power + spatial smoothness). **Methodological sources:** Dharmaprani et al. 2016 (the three-criterion detector itself); Whitham et al. 2007 (the paralyzed-subject EMG reference dataset against which the thresholds were tuned).
- `selector="both"`—IDE4EEG-side wrapper: runs **ICLabel + MNE detectors in parallel**, takes the **union** of their picks. In the GUI this is **both groupboxes ticked**; the greyed note in the "Components to remove." header makes the set-union semantics visible (∪ = union / sum; ∩ = intersection / overlap). TOML configs keep the canonical `"both"` string for back-compat. The Run-tab log line `ICA selector='iclabel ∪ find_bads': iclabel=N, find_bads=M, ∪=K (∩=…, iclabel-only=…, find_bads-only=…)` shows how the two detectors composed. `iclabel_min_prob` only gates the ICLabel half of the union—when the MNE detectors are the dominant contributor, sweeping `min_p` won't visibly change the union. Use `selector="iclabel"` if you want `min_p` to be the only knob driving the removal count.
- `selector="none"`—skip auto-detection entirely (only the decomposition runs); pick components yourself via the ✎ decision button, or use it when only ICA's *visualisation* is wanted.

6. **Manual review** (on-demand). A pipeline **Run never opens a review**—it is always pure `f(input, config)`. The ICA panel has a 👁 **eye** (before/after *signal* around the step) and a **decision button**: 🔒 views the component picks read-only, ✎ opens them editable (enabled by **Allow manual modifications**, Config tab). Clicking either button first **runs the pipeline up to the ICA step, forcing ICA to re-fit**, so the components you inspect match your current settings—on a long recording the re-fit takes a while, with its Run-log output, before the window opens. Editing then dispatches to Svarog (if the jar advertises `--select-mode ica_components`, showing the component sources + pre-rendered topomap PNGs, one per component) or `mne.preprocessing.ICA.plot_sources(...)` (MNE in-tree, GUI main thread) as a fallback; your selection **replaces** (does not union with) the auto picks and is **remembered for this recording in**

the current session (absolute-replace). A ✋ **marker** on the panel shows it is active; every later Run of the same file re-applies it until you revert (click ✋ → **Revert to automatic**). It is **not** written to the config.

> `selector` **choices.** `iclabel` (auto ICLabel), `find_bads` (MNE EOG/ECG/muscle reference detectors), `both` (their union—more aggressive automatic cleanup), or `none` (no auto picks; decomposition only). Any choice can be vetted with the ✏ decision button, whose reviewed set replaces the auto picks. **TOML back-compat:** the legacy `selector = "manual"` is migrated to `"none"` at load (`ide4eeg/config_schema.py:migrate_legacy_keys`); `"both"` is a *current* value, not migrated. The old per-surface `review_components` flag was removed—review is the decision button now.

7. **Apply.** `mne.preprocessing.ICA.apply(signal, exclude=bads)` reconstructs the full signal from the kept components only. **Code source:** MNE-Python in-tree (Gramfort 2014). The step returns the cleaned signal directly; `ica.py`'s job at this point is just plumbing the `exclude=` list.

Why "other" is not removed by default. The `other` class covers low-confidence components the classifier isn't sure about. Removing them by default would strip real brain activity the classifier happened to miss. Conservative default: leave `other` unticked (kept), and let the user opt into aggressive cleanup. The same reasoning applies doubly to `brain`—ticking it projects out neural signal rather than artifacts, so the panel shows an inline warning and the Run log warns as well; it is never blocked, since isolating an artifact subspace is a legitimate if unusual thing to want.

Parameter	Default	Description
`method`	`"picard"`	Decomposition algorithm: `"picard"` (upstream `python-picard`), `"infomax"` (MNE in-tree), or `"fastica"` (`sklearn.decomposition.FastICA`).
`selector`	backend-aware[1]	Auto-detection: `"iclabel"` (upstream `mne_icalabel`), `"find_bads"` (MNE in-tree EOG/ECG/muscle detectors), `"both"` (union), or `"none"`.
`manual_exclude`	*(absent)*	**Legacy / ignored.** Component-review decisions now live in the GUI session (RAM), not the config; this key is **stripped at config load** and never applied. Present only in old configs. Not a GUI-typed field.
`fit_highpass_hz`	`1.0`	Fit-time high-pass Hz (Winkler 2015). 0 = off. Applied to a `raw.copy()` only—analysis signal is untouched.
`fit_lowpass_hz`	`0.0`	Fit-time low-pass Hz. 0 = off (default). Applied to a `raw.copy()` only—drops muscle / HF harmonics from the ICA fit while leaving the analysis band wider.
`fit_notch_hz`	`0.0`	Fit-time notch frequency Hz. 0 = off (default). Applied to a `raw.copy()` only.
`fit_notch_harmonics`	`true`	When ON, the fit-time notch removes the fundamental + every integer harmonic up to Nyquist (50/100/150/… or 60/120/180/…). No effect when `fit_notch_hz = 0`.
`reject_uv`	`500`	Peak-to-peak µV—segments above this dropped from the pre-fit reject gate. 0 = off.
`reject_tstep`	`2.0`	Sliding-window length (s) MNE scores each segment against `reject_uv`. Forwarded to `ICA.fit(tstep=...)`.
`n_components`	`"rank"`	`"rank"` calls `mne.compute_rank(fit_copy)` (SVD-based) to defend against average-reference rank loss; int or float fraction of variance also accepted.
`max_iter`	`"auto"`	Maximum Picard/ICA fit iterations. `"auto"` lets MNE pick; raise it (e.g. `1000`) if the Run log warns the fit hit the iteration ceiling without converging.

Parameter	Default	Description
`iclabel_keep`	`["brain", "other"]`	Classes to **keep** when `selector="iclabel"`; everything else is removed. Note the GUI panel shows the complement (the classes to *remove*) and inverts on load/save—see the call-out above.
`iclabel_min_prob`	`0.0`	Min classifier confidence to trust the top-1 label—a probability from **0 to 1** (e.g. `0.8` = 80%; `0.0` = no minimum). A component whose top-1 probability is below this threshold falls into `"other"` (regardless of its actual top pick). With the default `iclabel_keep` containing `"other"`, low-confidence components are kept; with `iclabel_keep = ["brain"]` (no `"other"`), they are removed. **In GUI terms:** leaving `other` unticked (the default) keeps low-confidence components; ticking `other` removes them—that is the panel equivalent of `iclabel_keep = ["brain"]`. ICLabel's CNN softmax tends to be peaky (max-class prob often ≥ 0.9), so meaningful values are typically `0.9`–`0.99`. The Run-tab log line `ICLabel: removing N of M components; ...; max_p range [a .. b] (median c)` shows the actual distribution, so you can calibrate.
`find_bads_eog`	`true`	Run the EOG (`find_bads_eog`) detector. When on but `find_bads_eog_ch` is empty, EOG detection is skipped and logged at **INFO**—the ordinary state of an EOG-less recording (the GUI canonicalises the flag back to its default on such files for hash parity). Only a *configured* reference that turns out unusable (every named channel absent, or already marked bad) raises the end-of-run **WARNING**.
`find_bads_eog_ch`	`[]`	EOG reference channels for the `find_bads_eog` detector. Empty by default (opt-in): with a real EOG channel the GUI's EOG tick uses it directly; without one, the GUI populates a "choose proxy EOG:" picker from the frontal EEG channels **present in the file** (Fp*/AF*/F*) and your picks land here. No a-priori channel names are assumed.
`find_bads_ecg`	`true`	Also run `find_bads_ecg`.
`find_bads_muscle`	`true`	Also run `find_bads_muscle`.
`decim`	`1`	Subsampling factor for the fit (passed straight to `mne.preprocessing.ICA.fit`).
`random_state`	`42`	RNG seed.
`save`	`false`	🗄 drawer—writes `<base>-ica-cleaned-raw.fif`.
`save_components_audit`	`false`	"Save components plot and table"—writes audit tree under `artifacts_detection/ICA_EOG/`.

[1] **Backend-aware default.** The out-of-box `selector` is `"iclabel"` when an ICLabel backend (`onnxruntime` or `pytorch`) is importable, else `"find_bads"`—so a vanilla (backend-free) install runs without crashing. Install a backend (`pip install onnxruntime`) to get ICLabel as the default.

Failure modes.

- `reject` **gate drops every segment (or the fit otherwise fails) →** `ICAFailureError` (GUI: modal error; CLI/batch: logged ERROR, current file aborts, batch continues). Lowering `reject_uv` is the wrong direction here—*raise* it (or set 0 to disable) so usable segments survive the fit.

- **ICLabel flags every component** → `ICAFailureError` (GUI: modal error; CLI/batch: logged ERROR, current file aborts, batch continues).
- **Every class ticked for removal** (`iclabel_keep = []`) → `ICAFailureError` as above; the error names this cause first, since ticking everything is the easy way to reach it.
- **No class ticked for removal** (`iclabel_keep` lists all seven) → ICLabel classifies the components and removes none of them: the step runs but changes nothing. The panel shows "⚠ no class marked" and the Run log warns before the classifier runs.
- `find_bads` **detector raises** (wrong reference channel, version drift) → that detector's components are not removed; a prominent end-of-run WARNING names which artifact types (EOG/ECG/muscle) were skipped, so the "cleaned" label doesn't hide a retained artifact.
- **No ICLabel backend** with `selector="iclabel"`/`"both"` → caught at pre-launch (`hard.tools.iclabel_requires_backe` with the `pip install onnxruntime` hint; the runtime `ICAFailureError` is the final net for programmatic `run_file` calls that bypass the GUI gray-out + pre-launch check.

Output (when `save = true`). Under `<preproc_dir>/saved_steps/`:

- `<base>-ica-cleaned-raw.fif`—cleaned continuous signal. Use the 👁 button to view before/after.

When `save_components_audit = true`, under `<preproc_dir>/artifacts_detection/ICA_EOG/`:

- `<base>-ica.fif`—serialised `mne.preprocessing.ICA` object.
- `<base>_ica_classification.csv`—per-component table (index, class, probability, kept/removed).
- `<base>_ICA_topomap[_<n>].png`—component topomaps (`ica.plot_components`).
- `<base>_ICA_properties_<i>.png`—per-component property plots (`ica.plot_properties`).
- `<base>_ica_report.html`—bundled MNE Report.

Implementation status. IDE4EEG's `ica.py` is a workflow wrapper—it ships **no original numerical ICA code**. The dispatcher and state container (`mne.preprocessing.ICA`), the PCA pre-whitening, the rank estimator `mne.compute_rank`, the `find_bads_*` detectors, `ICA.apply`, and `ICA.plot_*` are all **MNE-Python in-tree** (Gramfort 2014). The decomposition kernels and the ICLabel classifier live in **separately released upstream packages**:

- `method="picard"` → `python-picard` (separate PyPI release; algorithm = Ablin, Cardoso & Gramfort 2018).
- `method="fastica"` → `sklearn.decomposition.FastICA` (algorithm = Hyvärinen 1999).
- `method="infomax"` → **MNE-Python in-tree** at `mne/preprocessing/infomax_.py` (algorithm = Bell & Sejnowski 1995; extended variant of Lee, Girolami & Sejnowski 1999 when `fit_params(extended=True)`).
- `selector="iclabel"` → `mne_icalabel` (separate PyPI release, MNE-affiliated but not core; classifier = Pion-Tonachini et al. 2019, Python port = Li et al. 2022). Inference backend = `onnxruntime` or `pytorch` (both optional, neither in MNE core).
- `selector="find_bads"` → **MNE-Python in-tree** (EOG correlation heuristic; ECG cross-trial phase per Dammers et al. 2008; muscle three-criterion per Dharmaprani et al. 2016, tuned against Whitham et al. 2007).

What IDE4EEG's `ica.py` itself contributes:

- The fit-time filter copy (`fit_highpass_hz` / `fit_lowpass_hz` / `fit_notch_hz` + `fit_notch_harmonics`) applied to `raw.copy()` only, never to the analysis signal—the default `fit_highpass_hz = 1.0` bakes Winkler et al. 2015's recommendation in as a default rather than as a hand-rolled user step. The harmonics chain for the fit-time notch mirrors the bad-channels preconditioning convention.
- The `reject_uv` ergonomic shim (single scalar µV → MNE's `reject=dict(eeg=...)` in V).
- The `n_components="rank"` default that calls `mne.compute_rank(fit_copy)`.
- The unified `selector` vocabulary (`iclabel` / `find_bads` / `both` / `none`) with a backend-aware default (`default_ica_selector()`) and a union-breakdown log line for `selector="both"`.
- The `iclabel_min_prob` confidence gate (components below the threshold are demoted to `"other"`, with a calibration log line showing the actual `max_p` distribution).
- The complete audit tree (saved-components CSV, topomap PNGs, properties PNGs, bundled HTML Report).
- The manual-review hook (Svarog `--select-mode ica_components` ↔ MNE `ica.plot_sources` dispatch based on what the installed Svarog jar advertises). The manual selection replaces (does not union with) the auto picks.
- The BLAS escape hatch (`threadpoolctl.threadpool_limits(limits=None)` around the fit).

- The failure surface—`ICAFailureError`, the pre-launch validator `hard.tools.iclabel_requires_backend`, GUI dialog wiring.
- Pipeline integration—`config_deps` for the staleness machinery, cancellation cooperation, parallelism inheritance, snapshot under `<base>-ica-cleaned-raw.fif` for the eye-button view.

References:

Ablin P, Cardoso JF, Gramfort A (2018) "Faster Independent Component Analysis by Preconditioning With Hessian Approximations." *IEEE Trans Signal Process* 66(15):4040–4049. DOI: 10.1109/TSP.2018.2844203.—Picard, the default `method=` algorithm; lives in upstream `python-picard`.

Bell AJ, Sejnowski TJ (1995) "An Information-Maximization Approach to Blind Separation and Blind Deconvolution." *Neural Comput* 7(6):1129–1159. DOI: 10.1162/neco.1995.7.6.1129.—Original Infomax; the algorithm behind `method="infomax"` (MNE in-tree).

Lee TW, Girolami M, Sejnowski TJ (1999) "Independent Component Analysis Using an Extended Infomax Algorithm for Mixed Subgaussian and Supergaussian Sources." *Neural Comput* 11(2):417–441. DOI: 10.1162/089976699300016719.—Extended Infomax; the objective Picard accelerates and what `method="infomax"` produces with `fit_params(extended=True)`.

Hyvärinen A (1999) "Fast and Robust Fixed-Point Algorithms for Independent Component Analysis." *IEEE Trans Neural Netw* 10(3):626–634. DOI: 10.1109/72.761722.—FastICA; the algorithm behind `method="fastica"` (lives in `sklearn.decomposition.FastICA`).

Hyvärinen A, Oja E (2000) "Independent component analysis: algorithms and applications." *Neural Networks* 13(4-5):411–430. DOI: 10.1016/S0893-6080(00)00026-5.—Canonical exposition of the ICA algorithmic family; cited for the **pre-whitening requirement** (step 3): all three `method=` choices require zero-mean, unit-variance, decorrelated input.

Winkler I, Debener S, Müller KR, Tangermann M (2015) "On the Influence of High-Pass Filtering on ICA-Based Artifact Reduction in EEG-ERP." *Proc. 37th Annual Int. Conf. IEEE EMBC*, 4101–4105. DOI: 10.1109/EMBC.2015.7319296.—The 1 Hz fit-time high-pass (`fit_highpass_hz`).

Pion-Tonachini L, Kreutz-Delgado K, Makeig S (2019) "ICLabel: An Automated Electroencephalographic Independent Component Classifier, Dataset, and Website." *NeuroImage* 198:181–197. DOI: 10.1016/j.neuroimage.2019.05.026.—The ICLabel CNN classifier behind `selector="iclabel"` (originally MATLAB/EEGLAB).

Li A, Feitelberg J, Saini AP, Höchenberger R, Scheltienne M (2022) "MNE-ICALabel: Automatically Annotating ICA Components with ICLabel in Python." *J Open Source Softw* 7(76):4484. DOI: 10.21105/joss.04484.—The Python port we actually import (`mne_icalabel.label_components`); the original ICLabel shipped only as a MATLAB/EEGLAB plugin.

Dammers J, Schiek M, Boers F, Silex C, Zvyagintsev M, Pietrzyk U, Mathiak K (2008) "Integration of amplitude and phase statistics for complete artifact removal in independent components of neuromagnetic recordings." *IEEE Trans Biomed Eng* 55(10):2353–2362. DOI: 10.1109/TBME.2008.926677.—Cross-trial phase statistics behind MNE's `find_bads_ecg`.

Dharmaprani D, Nguyen HK, Lewis TW, DeLosAngeles D, Willoughby JO, Pope KJ (2016) "A comparison of independent component analysis algorithms and measures to discriminate between EEG and artifact components." *EMBC* 825–828. DOI: 10.1109/EMBC.2016.7590833.—The three-criterion (slope + peripheral power + spatial smoothness) detector MNE's `find_bads_muscle` implements.

Whitham EM, Pope KJ, Fitzgibbon SP, Lewis TW, Clark CR, Loveless S, Broberg M, Wallace A, DeLosAngeles D, Lillie P, Hardy A, Fronsko R, Pulbrook A, Willoughby JO (2007) "Scalp electrical recording during paralysis: quantitative evidence that EEG frequencies above 20 Hz are contaminated by EMG." *Clin Neurophysiol* 118(8):1877–1888. DOI: 10.1016/j.clinph.2007.04.027.—Paralyzed-subject EMG reference dataset that the muscle-detector thresholds are tuned against.

Delorme A, Makeig S (2004) "EEGLAB: An Open Source Toolbox for Analysis of Single-Trial EEG Dynamics Including Independent Component Analysis." *J Neurosci Methods* 134(1):9–21. DOI:

10.1016/j.jneumeth.2003.10.009.—Background for `reject_uv` (peak-to-peak amplitude rejection as a standard pre-ICA step).

Onton J, Westerfield M, Townsend J, Makeig S (2006) "Imaging Human EEG Dynamics Using Independent Component Analysis." *Neurosci Biobehav Rev* 30(6):808–822. DOI: 10.1016/j.neubiorev.2006.06.007.—Canonical ICA-for-EEG best-practices review (motivates pre-ICA hygiene including `reject_uv`).

Gramfort A et al. (2014) "MNE software for processing MEG and EEG data." *NeuroImage* 86:446–460. DOI: 10.1016/j.neuroimage.2013.10.027.—The framework providing `mne.preprocessing.ICA` (dispatcher, PCA whitening, state container, `apply`, `plot_*`) and `mne.compute_rank`.

3.10 Gaze (video artifact detection)

GUI panel: Detect gaze artifacts Checkable, reorderable; **no drawer**—the marks `-tag.txt` is written unconditionally when the step runs, like the EEG-artifacts detector.

Identifies time intervals where the subject is not looking at the screen, using a synchronised video recording. These intervals are marked as `not_looking` artifacts and excluded from EEG analysis. Particularly useful for infant EEG studies where attention monitoring is essential.

The pipeline processes each video frame through two stages.

Stage 1: Face detection and identity (InsightFace). InsightFace is the only supported backend (`facetag_backend = "insightface"`).

Component	Model	What it does
Detection	RetinaFace	Finds all face bounding boxes in each frame
Identity	ArcFace	Computes a 512-d embedding per face, compared to reference photos
Tracking	Custom	Adaptive encoding + spatial/size/switch tracking across frames

What the reference photos actually define

The reference photos are not just "pictures of the subject" for identity—they set the **gaze baseline**. `check_gaze` computes a gaze vector for each frame and compares it to the vectors extracted from these photos; a frame counts as *looking* when the angle between them is below `facetag_max_angle` (default 20°). So the direction captured in the reference photos *is* the definition of "on task" for the whole recording.

That has one practical consequence worth getting right: **photograph the subject looking at whatever they are supposed to watch during the task**—the stimulus screen, or the fixation point—not at the camera. In a typical setup the camera sits above or beside the screen, so a camera-facing baseline is offset from the real task direction by that angle. Every genuinely on-task frame is then measured against the wrong reference, and correct behaviour gets scored as looking-away. The symptom is a run that marks far more gaze artifact than the video appears to justify, spread evenly rather than clustered at the moments the subject actually glanced away.

If the camera *is* the thing the subject watches, then camera-facing photos are correct—the rule is "look at the target", and the camera only happens to be it in some setups.

The same applies to the **Find frame** helper, which lets you grab reference shots from the video itself: pick a frame where the subject is attending to the task, not one where they happen to be facing the lens.

InsightFace also produces **5 facial keypoints** (left eye, right eye, nose tip, left and right mouth corners) as a byproduct of detection. These are reused by the InsightFace gaze backend.

Requirements: `pip install insightface onnxruntime`. Models (~200 MB) are downloaded automatically on first use to `~/.obci/ide4eeg/insightface/models/buffalo_l/`. (IDE4EEG redirects InsightFace away from its upstream default `~/.insightface/` so all of our runtime data lives under one tree; existing users with `~/.insightface/` get migrated once on first launch.)

Stage 2: Gaze estimation. Two backends, selected via `facetag_gaze_method`:

Backend	Config value	What it measures	Extra deps	Speed
L2CS-Net (default)	`"l2cs"`	Eye-gaze direction (where the eyes are actually looking; ResNet50, 3.92° MAE on MPIIGaze)	`l2cs` + PyTorch (~500 MB, CPU)	~20 ms / frame
InsightFace	`"insightface"`	Head pose only (where the head is pointing; reuses 5 keypoints)	None (reuses Stage 1)	~0 ms extra

The two backends are categorically different. L2CS-Net answers *"where are the eyes looking?"*. InsightFace answers *"where is the head pointing?"*. A subject whose head faces the camera but whose eyes are darting to a side monitor is "looking at the screen" by InsightFace's measure but not by L2CS-Net's. Pick L2CS-Net unless you have a specific reason to fall back.

Choosing a backend. Default `"l2cs"`—true eye-gaze direction is what most experimental designs want. Use `"insightface"` on **Intel Macs** (the only PyTorch wheel available there, 2.2.2, is broken against NumPy 2, so L2CS-Net cannot run—see the known-issues table below), when PyTorch can't be installed at all, or when L2CS-Net fails on a particular video (rare; e.g. very low-resolution face crops). For maximum throughput combine either backend with `facetag_frame_skip = 3-5`.

Parameters. All parameters are top-level (not in a TOML section).

Parameter	Default	Description
`prepare_video_artifacts`	`false`	Enable video artifact detection.
`video_path`	`""`	Path to `.mp4` video file. Auto-detected from EEG filename if empty.
`video_reference_dir`	`""`	Directory with 3–5 reference photos of the subject looking at their **task target** (the stimulus screen or fixation point)—*not* at the camera, unless the camera is what they watch. These photos define the on-task gaze direction; every frame is scored by its angle from them, so a camera-facing baseline biases the whole detection by the camera-to-screen offset. Empty = the GUI's **Reference faces** picker creates `<output>/IDE4EEG_OUT_<base>/preprocessing/reference_faces/`. **Batch (`--run`) caveat:** there is no picker—an empty or non-existent `video_reference_dir` makes the gaze step **silently skip** (a `WARNING` is logged and the run continues, no error), so set it explicitly for headless runs or gaze detection does nothing.
`video_exclude_dir`	`""`	Directory with photos of faces to exclude (e.g. mother). Empty = disabled.
`facetag_backend`	`"insightface"`	Face detection/identity backend. Only `"insightface"` is supported.
`facetag_gaze_method`	`"l2cs"`	Gaze backend: `"insightface"` or `"l2cs"`.
`facetag_max_angle`	`20.0`	Maximum angle (degrees) between current and reference gaze vectors to classify as "looking".
`facetag_gaze_smoothing`	`0.0`	Temporal smoothing (EMA alpha, 0–1). 0 = off; 0.3 = moderate; 0.1 = heavy.
`facetag_frame_skip`	`3`	Process every Nth frame (1 = every frame).

Parameter	Default	Description
`facetag_detection_skip`	`3`	Run face **detection** only every Nth processed frame (tracking fills the gaps); separate from `facetag_frame_skip`.
`facetag_downscale`	`1.0`	Resize factor before face detection (0.5 = half resolution).
`facetag_det_size`	`0`	InsightFace detection resolution. 0 = full frame; 640 = fast default.
`facetag_face_tolerance`	`0.6`	Maximum encoding distance for face match.
`facetag_min_interval`	`0.0`	Minimum duration (seconds) for a not-looking interval.
`facetag_no_face_is_artifact`	`true`	Treat frames with no detected face as "not looking".
`facetag_ref_use_roi`	`false`	Use face ROI (not full image) for reference photo gaze vectors.
`facetag_tracking_adapt_rate`	`0.5`	Rate of adaptive encoding update (0 = static, 1 = instant).
`facetag_position_weight`	`0.4`	Weight of spatial continuity in face tracking score.
`facetag_size_weight`	`0.2`	Weight of face size consistency in tracking score.
`facetag_switch_resistance`	`0.25`	Extra cost for switching to a different face.

Outputs. When `prepare_video_artifacts = true`:

- `<output>/artifacts_detection/video_artifacts/<file>-tag.txt`—Format-A MNE channel-scoped annotation file (`mne.Annotations.save`), one global `BAD_not_looking` mark per interval. The **same format the EEG-artifact detector writes**, so it loads with `mne.read_annotations(...)` and feeds Svarog through the combined marks overlay.
- `<video_stem>_facetag_<gaze_method>_review.json`—the review-state file, placed in the recording's **preprocessing output directory** (not next to the video); the Detect-gaze panel's 🔒/✏️ **decision button** opens the interactive **review window**, pre-populated from this JSON, where you can accept / edit the detected not-looking intervals. Unlike the bad-channel and ICA decision buttons, **this one does not re-run the detection**: when a review JSON exists it is reused as-is (the Run log says so), so the click is instant. It may still run a short truncated pipeline first, but only to refresh the EEG signal shown *behind* the intervals when no fresh upstream snapshot is on disk. If there is no saved gaze result at all, a 🔒 view refuses ("No gaze results yet") rather than starting the slow video pass—run the pipeline once, or turn on **Allow manual modifications** so ✏️ can compute and edit here. Gaze is a marking step with no signal delta, so it has no 👁 eye—its only header button is the decision button, and ✏️ (master switch on) **remembers** your reviewed intervals for this recording in the current session (absolute-replace); a ✋ **marker** appears and every later Run of the same file re-applies them (video pass skipped) until you revert. They are **not** written to the config. 🔒 views them read-only.
- `BAD_not_looking` MNE annotations carried on the signal carrier and written to a Format-A `-tag.txt` (`artifacts_detection/video_artifacts/<file>-tag.txt`)—Svarog and MNE viewers render them as colored overlays. Only the **EEG-artifacts** detector keeps a 👁 eye (it overlays its `-tag.txt` marks on the preceding step's signal in Svarog); the gaze step uses the decision button above instead. They are two distinct reorderable steps and do not share a snapshot.

The Run-tab **Stop** button cancels the video frame loop cooperatively—`cancel_event.is_set()` is checked once per frame inside `facetag_video`, so a multi-hour video doesn't keep running after Stop.

Speed tips.

- **Frame skip** (`facetag_frame_skip = 3-5`): ~3–5× speedup at 30 fps with minimal accuracy loss.

- **Downscale** (`facetag_downscale = 0.5`): ~2× speedup per frame. Avoid below 0.3 for small faces.
- **Detection size** (`facetag_det_size = 640`): faster than full-frame (0) but may miss distant faces.
- **Trim the window** (a `trim` step *before* Gaze in `step_order`): facetag only decodes the trimmed `[trim_start, trim_end]` span of the video, not the whole file—a large saving on long recordings. Note this bound is set by the **trim step actually running**: it only applies when `trim` is present in `step_order` (the step is what writes the runtime window facetag reads). Setting `trim_end` in the config *without* a `trim` step leaves the scan unbounded (the full video is processed).

These options combine multiplicatively.

3.11 Detect EEG artifacts

GUI panel: Detect EEG artifacts (checkable, reorderable, on by default). The EEG-artifact detection suite: a robust-statistics, multi-scale family of detectors (`p2p`, `slope`, `hf`, `flat`) that mark contaminated spans for the downstream Mark detected artifacts stage. It has **no 🖫 drawer**: the per-channel marks `-tag.txt` is the step's primary output and is written unconditionally when the step runs (a marking step has no signal snapshot to gate).

This step does not modify the signal. Each detector *marks* artifacts as a channel-scoped `BAD_<type>` carrier on the signal (MNE annotations, with the robust-z folded in as `BAD_<type>|z=…`), and writes a per-channel `eeg_artifacts/<file>-tag.txt` review file (MNE's channel-scoped annotation `.txt` format)—written unconditionally when the step runs, since the marks file is the step's primary output. It leaves the signal and segments intact, in either segmentation mode. Automatic rejection is a later phase.

p2p and **slope** are **per-sample** detectors that share a detection frame and run on a dyadic sequence of window widths `L = 2, 4, 8, …` samples (octave-spaced), derived per sampling rate from physical-millisecond constants. Each scale is robust-z'd over the whole recording (median/MAD) and the scales are combined by max-z. The ladder is split at the crossover (50 ms): the **slope** detector owns the rungs below it (the steep band), **p2p** the rungs above (the amplitude band), up to **max scale** (300 ms; slower events are the drift detector's / analysis high-pass's job). **hf** is a windowed band-power detector on its own high-frequency band—a different method for a different artifact class (sustained muscle/EMG rather than transient steps and pops). **flat** is the low-amplitude counterpart of p2p—the *low tail* of the same peak-to-peak feature, detecting transient flat/dropout spans.

- **p2p—peak-to-peak amplitude.** A per-sample running max − min over the amplitude band; flags windows whose peak-to-peak is unusually large *for the channel*—blinks, gross movement, the sustained level of a pop—as `p2p` tags. One-sided (large only).
- **slope—regression slope.** A per-sample least-squares regression slope, reported in µV/ms over the steep band, down to the 2-sample first difference; flags fast **steps**—electrode pops, saccadic steps, single-sample jumps—as `slope` tags. Two-sided (up and down).
- **hf—muscle (EMG) band power.** Band-passes the copy to a high band (default `[90, min(250, 0.45·fs)]` Hz, where scalp power is mostly muscle), takes a robust power estimate per **0.5-s non-overlapping window**, robust-z's those window powers over the whole recording (per channel), and flags windows whose `abs(z)` exceeds the threshold as `hf` tags. Needs `fs ≥ 200 Hz` (the upper edge is auto-capped below Nyquist); the step is skipped with a warning when the band would collapse.
- **flat—relative flat span.** The low-amplitude counterpart of p2p: a window is flat when its running peak-to-peak falls below `flat_fraction × median(p2p)`—the channel's *own* median, so the test is relative and adapts to each channel's amplitude (no absolute µV threshold, so it does not misfire on bipolar / low-gain / pediatric montages). Contiguous flat runs `≥ flat_min_duration_s` (default 0.5 s) are emitted as `flat` tags—transient disconnect / saturation / dropout. **Marks the span, not the whole channel** (that is the [Detect bad channels] step's job), so a channel flat for one window but healthy elsewhere keeps its good data. Assumes flat is a *minority* of the recording (a mostly-flat channel is the bad-channel detector's job).

Shared detection frame. All four run on a throwaway EEG-only copy with a mains notch only (50/60 Hz + optional harmonics—the full comb matters for hf, which spans the harmonics; no high-pass—p2p is DC-invariant and the slope is its own high-pass; **no low-pass**—it would smooth the sharp structures and erase the hf band). p2p, slope and flat read the wide notched signal directly (flat from its running peak-to-peak); hf additionally band-passes that copy to its high band. All emit one channel-scoped annotation (and one `-tag.txt` row) per contiguous flagged run per channel, with no padding or merge-gap—the run *is* the detection; extent comes from the multi-scale / multi-window response and final decisions are deferred to the aggregation/review stage. (The slope detector blanks a boundary region of width ≈ half the coarsest slope rung at each end of the recording, where the convolution edge would otherwise create phantom tags.)

Parameters (the panel)

Field	Default	Meaning
Notch	`50`	mains-notch frequency (off / 50 / 60 Hz), plus **harmonics**
Robust-z (p2p / slope / hf)	`5.0` / `5.0` / `5.0`	flag a scale/window where `abs(z)` exceeds it—a **separate** field in each detector's box
flat (ptp/median <)	`0.2`	flat when a window's peak-to-peak drops below this fraction of the channel's median ptp
adaptive floor: k = N × median p2p	`1.0`	**default** p2p/slope amplitude floor, relative to each channel's own median p2p (`floor_rel_k`); uncheck → fall back to the absolute µV floor below (`floor_rel_k = 0`)
absolute floor	`50` µV	deflection floor + too-clean-channel guard, used only when the adaptive floor is off (`floor_rel_k = 0`)
p2p	on	run the peak-to-peak amplitude detector
slope	on	run the regression-slope detector
muscle (hf)	on	run the high-frequency muscle/EMG band-power detector
flat	on	run the relative flat-span detector (low tail of p2p)

Config-only constants (in `[eeg_artifacts]`, no GUI field—physics/expert constants): `crossover_ms` (50), `max_scale_ms` (300, the coarsest p2p detection window), `amp_floor_window_ms` (0 = inherit `max_scale_ms`; the window over which the adaptive amplitude floor is estimated, decoupled from the detection cap), `slope_min_samples` (2, the first-difference floor), `min_windows` (20, the `n_eff = n / L_coarsest` stability gate—applied to both dyadic detectors, each at its own coarsest rung, so p2p with the coarser rungs trips first; **set 0 to disable the guard**—the in-panel 20/40 advisory is fixed and does not track this value—see the window-count feedback below), the muscle band/window: `hf_lo_hz` (90), `hf_hi_cap_hz` (250, upper-edge cap before the `0.45·fs` Nyquist clamp), `hf_window_s` (0.5); and the flat-span windows: `flat_window_s` (0.2, the ptp window for flatness), `flat_min_duration_s` (0.5, the shortest flat run worth marking).

The amplitude floor is adaptive (per recording). "Floor: k = N × channel median p2p" (`floor_rel_k`, default **1.0**) sets the p2p and slope amplitude floor *relative to each channel's own amplitude*: `floor = floor_rel_k · median_ptp`, where `median_p2p` is the per-channel median of the running 300 ms peak-to-peak (the same `max-min` scale the `flat` detector uses). This tracks each recording's gain/montage instead of a fixed µV value, which keeps detection aggression comparable across recordings of very different amplitude—a fixed µV floor over-suppresses low-amplitude ERP (`median_p2p ≈ 20 µV`) and is a near no-op on high-amplitude recordings (`median_p2p ≈ 66 µV`). Set `floor_rel_k = 0` to use the *absolute* µV floors below instead.

The absolute floor doubles as a guard (legacy path, `floor_rel_k = 0`). "Amplitude floor = N µV" (`min_amplitude_uv`) sets the smallest p2p deflection worth tagging *and* floors the robust scale so a near-flat or dead channel cannot blow up its own z. Used only when `floor_rel_k = 0`; set it as a deflection height in µV, or `0` to flag on robust-z alone.

Why the slope backstop is a µV step, not a µV/ms slope. The slope detector reads a least-squares regression slope over each dyadic window `L`; for a step of height `h`, that reading is `h · P(L) · fs/1000` µV/ms, where `P(L) = Σ_{k>0} c_k` is the kernel's *step-response gain* (the slope a unit step produces at scale `L`; `P(L) ≈ 1.5/L`, computed exactly from the kernel). A short window sees a step as steep, a long window "dilutes" it—so the *same* step reads a different µV/ms at every scale. The backstop is set to `A · P(L) · fs/1000`—exactly the slope an `A`-µV step would produce at `L`—so the flag test `|slope| > backstop` reduces to `h > A`: the `P(L)·fs/1000` cancels, and one µV step amplitude gives a **consistent, scale- and fs-invariant** threshold across the whole bank. A raw µV/ms number could not: it would mean a different step size at every rung and drift with the sampling rate.

The amplitude ceiling (Advanced, off by default). Its symmetric partner: "Amplitude ceiling = N µV" (`p2p_ceiling_uv`, `0` = off) *also* flags any p2p window above an absolute µV level, regardless of robust-z. A channel whose amplitude is uniformly huge has its MAD scale *with* it, so the per-channel robust-z never fires—only an absolute ceiling catches that gross-excursion class. It operates on the p2p detector's **sliding dyadic windows**

(`crossover_ms…max_scale_ms`, ≤300 ms), *not* the whole segment—so it catches **fast / local** gross excursions; a slow whole-segment swing (large only over tens of seconds) is the high-pass / drift detector's job, by design.

Window-count feedback. The panel warns when the trimmed signal is too short for a stable whole-record reference at the coarsest p2p scale (`n_eff = n / L`): **20–40 effective windows** → an orange advisory; **under 20** → a red message and p2p is switched off.

Full specification, literature, and design rationale: artifact-detector review.

3.12 Mark detected artifacts

GUI panel: Mark detected artifacts—a checkable header whose tick is a pure indicator that mirrors the detectors, not a control you set. It is disabled (cannot be ticked) when neither *Detect gaze* nor *Detect artifacts* step is on, and turns on the moment either of them is enabled. Toggling it by hand has no effect.

`[Mark detected artifacts]` owns no detector (it never decides what an artifact *is*—that is settled upstream), but it does two things, and only the second is dropping. - It **conditions** the single channel-scoped **carrier** (the `BAD_<type>` marks the Detect-EEG-artifacts `p2p` / `slope` / `hf` / `flat` detectors and gaze attach) into the final marks—per-type floor / consolidate / dilate (the "Adjust time spans of marked artifacts" panel below). These conditioned marks are the one product every consumer reads: the gray overlay band, the continuous-mode PSD omission, EEG-profile atom rejection, and the segment drop—so they all agree by construction. This conditioning runs regardless of whether you drop, and is the panel's primary work in continuous mode (where the drop is off by default). - It optionally **drops** segments scored against those marks via the **"Drop marked"** checkbox. It runs no detection of its own: the absolute amplitude ceiling lives on the `p2p` detector (`p2p_ceiling_uv`) and flat spans on the `flat` detector, both in §3.11.

The three artifact-rejection switches

Detecting artifacts and *acting* on them are separate. The Detect EEG artifacts and Detect gaze artifacts steps decide **which marks exist**; they never drop or omit anything themselves. Each analysis type then has its **own** rejection control, acting on its **own** data representation, and the three are independent—turning one on or off does nothing to the other two:

Analysis / representation	Rejection control (exact GUI label)	Panel	Effect when ON
Epochs / ERP-type—the butterfly / image / GFP / joint / topomap ERPs, TFR, Comparison, per-segment stats, the saved `*_clean-epo.fif`	**"Drop marked"**	Mark detected artifacts (this panel)	Removes the marked epochs. **Unchecked → mark-only:** the pipeline still cuts, saves, and computes stats—nothing is dropped, and the marks are still shown.
EEG profiles (§4.2)	**"Remove atoms in artifact-marked spans"**	EEG Profiles	Drops MP atoms whose time-center (t0_s) falls in one of this channel's conditioned artifact spans—per-channel, so a Cz artifact never removes an Fp1 atom; the whole-montage union is only a fallback for a book-only band—before binning.
MMP → dipole sources (§4.1)	**"Remove atoms in artifact-marked spans"**	MMP → dipole sources	Omits macroatoms whose time-center falls in the conditioned whole-montage artifact band, before dipole fitting.

The EEG Profiles and MMP-dipole switches carry the **same label** ("Remove atoms in artifact-marked spans") because both act on MP atoms, but they are two independent checkboxes on two

panels. In **continuous (rest)** mode the PSD spectra omit marked spans via a fourth, separate switch—`[segmentation].reject_by_annotation` (§4.5)—not "Drop marked".

A detector checkbox and "Drop marked" are different controls. Unchecking a detector ([Detect EEG artifacts] or [Detect gaze artifacts]) means *"stop marking that artifact type"*—it removes marks from the pool. "Drop marked" means *"act on whatever marks exist and remove the contaminated epochs"*. Unchecking a detector thins the marks every switch above reads; unchecking "Drop marked" leaves the marks intact but stops the epoch drop. The **Mark detected artifacts** header tick is neither of these: it is a **pure indicator** that auto-follows the detectors (off and greyed when neither detector is on) and has no effect on the pipeline when you click it.

Recipes.

- **Reject gaze artifacts but not EEG artifacts.** Uncheck [**Detect EEG artifacts**]; keep [**Detect gaze artifacts**] on and **"Drop marked"** ticked. Only `not_looking` marks remain, so only gaze-contaminated epochs drop.
- **See EEG profiles *without* artifact rejection.** Uncheck **"Remove atoms in artifact-marked spans"** in the **EEG Profiles** panel—**not** "Drop marked" (that governs epochs, not the atom profile). The profile is then a pure function of the MP book.
- **Compare with-rejection against no-rejection.** Use **two separate configs / output trees**. In the no-rejection arm turn **"Drop marked"** off *and* leave the epoch review empty (see When a manual review resets), so no remembered decision carries a drop across from the with-rejection arm. For a profile or dipole comparison, likewise toggle **"Remove atoms in artifact-marked spans"** between the two arms.

What it *does* own is its **aggregation policy**—the knobs that decide how the consolidated marks collapse into a drop. The policy reads as one line: *"epoch bad in 1 channel if artifacts > N % or > M ms; whole epoch bad if > P % of channels bad"*. There are no enable checkboxes—a `0` in any field disables that gate (the field is the toggle).

A channel is bad when its **contaminated-sample percentage** exceeds `artifact_threshold` (the `%` field; shown as a %, stored as the `[0,1]` fraction). `0` = this gate off (a `0 %` threshold under strict `>` would otherwise mean "any contamination", so `0` maps to off, not threshold 0), *or* when its **longest contiguous contaminated run** reaches `artifact_min_run_ms` (the `ms` field); `0` = off (fraction-only). G1 exists because the fraction rule never fires on a brief, sharply-localised mark in a long window—a 0.5 s burst is only 2.5 % of a 20 s rest window.

The epoch drops only when more than `consensus_frac` of the channels are bad. The field default is `0` = `>0 %` = *any* single bad channel condemns the epoch; raise it so a single transiently-noisy electrode no longer discards an otherwise-clean montage.

The GUI derives `enable_fraction` / `enable_run` from each field being `> 0`; a config that disables a gate by flag is shown with that field at `0`.

eye button (mode-aware). In **continuous** mode with the window-drop off (mark-only—the default), the eye opens a **layered marks overlay** in Svarog rather than a segment review: a gray *conditioned* band (the floor→consolidate→dilate union the spectra omit) with the per-channel **colored source marks** (p2p / slope / hf / flat / gaze) superimposed on top. The **Show detailed source marks** checkbox (continuous mode) sets whether that colored layer is shown by default—it stays toggleable live inside Svarog. In **event-locked** mode, or **continuous with "Drop marked"** ticked, the eye opens the **whole-segment review** instead (windows are the unit being dropped there). The layered overlay needs a Svarog jar that implements the `--marks-*` layering flags; older jars show a flat overlay.

Adjust time spans of marked artifacts (a disclosure below the gates)—three optional per-type controls (`p2p` / `slope` / `hf` (EMG) / `flat` / `gaze`), each gated by its checkbox, applied in the order **floor → consolidate → widen** and shared by the epoch drop *and* the continuous-mode PSD omission (§4.5): **at least N ms** (`min_mark_ms`, a duration floor that widens marks shorter than N—rescues e.g. a 2-sample slope no gate would catch); **consolidate gaps < N ms** (`consolidate_ms`, morphological *closing*—merge two same-type marks whose clean gap is shorter than N; pre-fills the field-standard 100 ms, mirroring MNE's `min_length_good`); and **each side + N ms** (`dilate_ms`, a symmetric pad)—for the z-carrying detectors (`p2p`/`slope`/`hf`) the pad scales per mark by `× (z - z_thr + 1)`, so a stronger artifact pads proportionally wider (`flat`/`gaze` carry no z → flat pad).

Parameter	Default	Description
`auto_reject`	`true`	**Event-locked** drop toggle—the **"Drop marked"** checkbox. Checked → drop the marked epochs; **unchecked → mark-only** (cut/save/stats run, nothing drops). Ignored in continuous mode—see `drop_in_continuous`.
`drop_in_continuous`	`false`	**Continuous** drop toggle—the **"Drop marked"** checkbox in continuous mode, off by default (dropping fixed windows breaks the signal's continuity ⚠). The PSD spectra (`reject_by_annotation`), EEG profiles (via the conditioned artifact band), and MMP dipole fitting (via `[dipole_fitting].reject_artifacts`, §4.1)—all reading the same whole-montage conditioned band—omit marked spans without dropping; the remaining consumers—connectivity, TFR, per-segment stats, the saved `*_clean-epo.fif`—run on the cut windows (`rest_clean`) and see contaminated data unless this is on. The panel's epoch-drop criteria are hidden until "Drop marked" is ticked.
`artifact_threshold`	`0.3`	Per-channel contaminated-sample fraction (0–1) that makes a channel bad; the GUI shows it as a percent (30 %). `0 %` = gate off. Lower = stricter.
`artifact_min_run_ms`	`200.0`	G1 run-length trigger (ms), OR-ed with the fraction rule. `0` = off (fraction-only).
`enable_fraction`	`true`	Derived from the `%` field being `> 0` (no checkbox); `0 %` writes `false`.
`enable_run`	`true`	Derived from the `ms` field being `> 0` (no checkbox); `0 ms` writes `false`.
`enable_consensus`	*(absent)*	**Legacy / removed.** The quorum is always applied, and its test is strict, so `consensus_frac = 0` already *is* the any-channel rule this flag's `false` selected. **Migrated at config load, not ignored:** `enable_consensus = false` sets `consensus_frac = 0.0` and warns—because a `false` flag alongside a non-zero `consensus_frac` (which was inert while the flag was off) would otherwise become a real quorum and silently drop fewer segments. `enable_consensus = true` is simply removed.
`consensus_frac`	`0.0`	Quorum: drop only when more than this fraction of channels are bad (GUI shows a percent). `0` = `>0 %` = any single bad channel.
`min_mark_ms`	all `100`	Per-type `{p2p, slope, hf, flat, gaze}` duration floor (ms) widening short marks.
`dilate_ms`	`20`; `flat/gaze` `50`	Per-type symmetric pad (ms); for `p2p`/`slope`/`hf` it scales per mark by `× (z − z_thr + 1)`, so their base can be smaller—`flat`/`gaze` carry no z (fixed pad) and default wider.
`consolidate_ms`	all `100`	Per-type gap-closing (ms): merge same-type marks whose clean gap is shorter than this (applied only when its per-type guard is on).
`consolidate_enable`	all `true`	Per-type guard for `consolidate_ms` (the "consolidate gaps" checkbox); on by default.

The TOML section is `[segment_rejection]`; the keys above are its policy—the span-conditioning knobs, the per-mode "Drop marked" toggle, and the fraction/run/consensus drop gates.

Manual segment review

A pipeline **Run never opens a review**—it is pure `f(input, config)`. The Mark detected artifacts panel exposes the epoch review through two header buttons:

- **eye** (`_view_mark_artifacts_step`)—a **read-only**, mode-aware view. It triggers a one-off preprocessing-only run (analyses skipped) terminating at this review, so you can inspect the rejections without committing

to a full Run. In continuous mark-only mode it opens the layered marks overlay; in event-locked mode (or continuous with window-drop on) it opens the whole-segment review.

- 🔒/✏ **decision button** (`_edit_step_result("segments")`)—the **editable** review. With **Allow manual modifications** on (Config tab) it reruns preprocessing, opens the epoch browser, and **remembers** your reviewed rejections for this recording in the current session (absolute-replace of exactly the reviewed epochs). A ✋ **marker** appears and every later Run of the same file re-applies them until you revert. They are **not** written to the config. With the master switch off it is a 🔒 read-only view.

Both use the same Svarog → MNE dispatch as bad-channels and ICA: if the installed jar advertises `--select-mode bad_epochs`, Svarog opens the bad-epoch panel; otherwise MNE's `epochs.plot(...)` runs on the GUI main thread.

What the review window shows. The browser does not display the original continuous recording—it shows your selected epochs **concatenated end-to-end into one signal**, in event-time order. With `start_offset = -0.2`, `stop_offset = 1.0`, each epoch is a 1.2 s block, so 720 selected epochs become an ~864 s review signal that exists only for this step. That is also why the right-hand list (Svarog: *Odcinki do odrzucenia*, "segments to reject") reads `#index start-end s` with times on this **concatenated** axis, not the recording's clock—`#0 -0.20-1.00 s`, `#1 1.00-2.20 s`, and so on. The list is every epoch, not a list of already-rejected ones; a row marked for rejection is shown struck-through in red with `(odrzucone)`.

The vertical lines delineate the epoch structure on that concatenated axis—epoch boundaries and each epoch's locking-event (stimulus) onset, which sits `+|start_offset|` into the block. (The temporary review `-epo.fif` also carries the recording's original event tags, including event types you did not select, e.g. `tent`—these are metadata, never rejection marks.) They are *not* artifact flags: at this point nothing has been auto-rejected unless the artifact aggregator dropped epochs upstream (see §3.12).

To mark/keep: double-click a row, or Alt-click the signal under the cursor, to toggle an epoch's rejection; the selection is saved when you close the window. Closing without marking keeps every epoch. Two Svarog 4.20 ergonomics to be aware of: the time axis is unlabeled, and the page width can open close to one epoch—widen **time scale** so epoch boundaries sit inside the view rather than at its edges. Clearer epoch numbering and a labeled per-epoch timeline are requested upstream.

When a manual review resets

A remembered manual review (bad channels, ICA components, epochs, gaze) is tied to the exact upstream signal it was made against. It lives in the current **session**—in memory, for the loaded recording—is re-applied to every later Run of that file, and is shown by a ✋ **marker** on the step (and listed on the Run tab). It is **not** written to the config; the reproducible record of a reviewed result is the **saved cleaned signal**, not a config key.

A remembered decision goes away in exactly these ways, and never silently:

- **You revert it.** Click the ✋ marker → **Revert to automatic** (or **Revert all to automatic** on the Run tab). The step returns to the automatic result.
- **You change something upstream of it.** Because the decision was made against a specific signal, editing an upstream step (channel selection, trim, resample, reference, filters, ICA, or the detector / scoring knobs the review sat on) would make it stale. IDE4EEG never applies—or silently drops—a stale decision: the moment your edit invalidates one, a dialog offers **Discard & change** (drop the review, keep your edit) or **Cancel change** (undo the edit, keep the review). A **Run** re-checks as a safety net (**Discard & run automatic** / **Cancel run**).
- **You load a different recording, or close IDE4EEG.** Session memory is per-file and is not saved to disk.

Notes:

- **A review is remembered only if you actually changed it.** Opening the editable review (the ✏ decision button, with **Allow manual modifications** on in the Config tab) and closing it **unchanged** remembers nothing—the run stays fully automatic. Deliberately un-marking everything and accepting is itself a decision ("reject nothing"), which *is* remembered—distinct from not reviewing at all.
- **The Run log names a re-applied decision** so you can tell a reviewed drop from an automatic one, e.g. `Segments: re-applying 3 remembered manual rejection(s) from this session.`

Automatic postamble outputs

After the reorderable panels and the Mark detected artifacts stage, the pipeline finalises the run automatically. These steps have no GUI panel of their own—they always run.

Save segments. On a full run the pipeline writes the final cleaned epochs as `<base>_clean_<mode>-epo.fif` (`<mode>` = `epochs` or `rest`, e.g. `subject01_clean_epochs-epo.fif`) to `preprocessing/saved_signals/`—the primary preprocessing output, never suppressed by save toggles. (View-only eye-button runs skip the write, and a run that yields zero clean epochs removes any stale file so on-disk epochs never lie.) In **event-locked** mode every analysis runs on these epochs. In **continuous (rest)** mode the cut windows still feed connectivity, dipole, TFR and per-segment statistics, but the **PSD spectra estimate on the *continuous* preprocessed signal** (omitting artifact-marked spans via `[segmentation].reject_by_annotation`, default `true`—Mode: continuous (rest), §4.5), and **EEG profiles read the MP-book atoms** (omitting artifacts via `[eeg_profiles].reject_artifacts`) rather than the cut epochs.

Per-segment statistics. Computes per-epoch per-channel statistics (id, tag, channel, t_start, t_stop, bad-segment / bad-channel flags, and one `<detector>_arts` column per artifact detector giving that channel's artifact fraction) into a pandas DataFrame. This table is currently built and passed to the analysis stage but is not yet written to disk or consumed by any analysis—there is no `stats/` CSV for it.

MP book reuse

MP decomposition follows a hash-controlled reuse policy: the freshness check runs whether or not `mp_decomposition` is in `step_order`.

Preprocessing tab → "MP Decomposition" checkbox	Book on disk	Hash	What happens
Checked	none	—	Run empi, write `book_*.db`, stamp the current config's hash.
Checked	present	matches	**Skip empi**, reuse the existing book. Log: `MP Decomposition: reusing existing book (hash match)`.
Checked	present	differs / legacy	Run empi, overwrite the book with a fresh hash.
Unchecked	none	—	Skip MP; downstream MP-consuming analyses fall back to ephemeral per-ERP decomposition.
Unchecked	present	matches	Reuse the book silently.
Unchecked	present	differs	**Halt** with an error explaining how to recover (re-check the box, or restore the upstream parameters that produced the book).
Unchecked	present	legacy (no hash)	Reuse with a one-time soft warning.

This means iterating on **filter-criteria parameters** (the `[matching_pursuit.filter]` sub-block—a legacy, unused sub-block nothing in `analysis/` currently consumes, distinct from the MP-filter step's own `[mp_filter.filter]`) does not trigger an empi rerun—those criteria operate on already-decomposed atoms, so the sub-block is stripped from the book hash. Same for runtime-only fields (`cpu_workers`, `outputs`).

The hash covers everything that determines what data MP sees: the source EEG file's identity (name + size + mtime), `preprocessing.step_order` (so reordering invalidates), and every config section read by a step that runs before MP—including `filters`, `ICA_EOG`, `montage`, `bad_channels`, `choosing_channels`, and the decomposition-relevant fields of `matching_pursuit` (algorithm, channels, iterations, `explained_energy`, optimisation, …). Final-segment-handling sections (`epochs`, `mp_filter`) are excluded—they can't influence MP input.

> **MP filter has its own internal book.** Same hash-controlled reuse policy as above, but scoped to the `[mp_filter]` block (with `filter` sub-block stripped). Editing the internal decomp params or `window_length` invalidates and re-runs empi; editing reconstruction criteria never invalidates. See §3.7.

Consumer-channel check on reuse. Even when the hash matches, the on-disk book's `channel_names` list may not include a channel that an enabled downstream consumer needs (`eeg_profiles.channel`). When that happens the analysis's pre-run validation halts with a structured error pointing at two recovery paths: set the analysis channel to one in the book, or check MP Decomposition AND update `matching_pursuit.channels` to include the required channel (or `"all"`)—the latter rewrites the book.

The same rule applies to `ide4eeg --run config.toml` and to scripts exported via `api.generate_script`. MP enable state lives in `preprocessing.step_order`, not in any top-level flag. Both **Run Pipeline** and per-analysis **Run** buttons respect the MP checkbox identically—checking it forces a recompute, unchecking it lets auto-discovery decide.

> **Migrating from older configs**: the legacy `prepare_mp_decomposition` TOML key was removed. To enable MP, add `"mp_decomposition"` to `preprocessing.step_order`. Configs that still use the old key log an error and run without MP.

---

4. Analysis tab

The Analysis tab groups all post-preprocessing computations: ERPs, time-frequency analyses, statistical tests, connectivity, source localisation, and the external MNE-Python catalog. Each analysis has its own checkbox, parameter group, and per-analysis ▶ Run button. Three IDE4EEG-native analyses appear at the top, then a separator, then a series of MNE-Python wrappers grouped by category (ERP / Spectra / Time-frequency / Spatial / Comparison / Inverse solutions).

Run buttons & preprocessing-skip logic

What "Run" means per analysis

Some analysis sections (Connectivity, EEG Profiles, MMP Dipole Fitting) have their own ▶👁 run-and-view button. Clicking it narrows the run to just that analysis—every other `prepare_*` flag is forced off—and otherwise behaves like Run Pipeline. The current GUI state is collected into a config dict at the moment you click; a `config.toml` reflecting the per-analysis narrowing is saved to the output directory so you can see exactly what was executed.

Skipping preprocessing

What a per-analysis [Run] saves depends on whether the analysis is *book-only* (reads the MP book directly) or not.

Analysis	Per-analysis [Run] behaviour
Book-only: EEG Profiles, MMP Dipole Fitting	If a matching MP book is on disk, skips signal loading and preprocessing entirely—reads atoms straight from the book and runs only the analysis. Fast path. **Caveat for EEG Profiles:** it takes the fast path only when artifact rejection is off (`[eeg_profiles].reject_artifacts = false`) or a *fresh* artifact band is already on disk; with the default `reject_artifacts = true` and no fresh band it re-runs preprocessing to regenerate the artifact carrier even when a matching book exists. Dipole fitting is always book-only. If no matching book exists, it falls through to the full pipeline.
Non-book-only: Connectivity, MNE-catalog ERP / Spectra / Time-frequency / Spatial / Comparison / Source-estimation panels	Runs **full preprocessing** (same as Run Pipeline), then only this analysis. Saves the cost of *other* enabled analyses; no faster than Run Pipeline if this is your only enabled analysis.

Notes that apply to both cases:

- The MP-decomposition checkbox on the Preprocessing tab still controls recomputation—checking it forces empi to re-run; unchecking it lets the analysis's pre-run validation reuse an existing book if its hash matches the current config (see MP book reuse).
- If the analysis requires the standalone MP book (Profiles or Dipole with the default `mp_book_source = "decomposition"`) and none exists, the GUI's preflight either auto-adds `mp_decomposition` to `preprocessing.step_order` (when an empi binary is installed) or refuses with a clear message. MP filter is NOT a standalone-book consumer—it runs its own internal MP decomposition (see §3.7), so enabling it doesn't trigger the auto-add. Profiles / Dipole configured with `mp_book_source = "mp_filter"` need mp_filter in the step order instead.

Event-dependent vs continuous-signal analyses

Some analyses require event markers (≥ 2 event types are needed to compute differences); others work on continuous data. The Analysis tab marks each analysis with which modes it supports:

Analysis	Available for
MMP → dipole sources	Both modes
EEG profiles	Both modes
Connectivity	Both modes
MNE catalog (ERP / Comparison)	Needs events
MNE catalog (Spectra / Spatial / Time-frequency)	Both modes (ERD/S topomap and Evoked topomap series are event-locked-only)
MNE catalog (Inverse solutions)	Needs events

When the loaded recording's segmentation mode doesn't supply what an analysis needs, the GUI greys out the unavailable entries and shows a tooltip explaining why.

4.1 MMP → dipole sources

GUI panel: MMP → dipole sources—checkable · ▶👁 run-and-view · **?** help · ⚠ **needs MMP1 book** until one is on disk **Config section:** `[dipole_fitting]` **Prerequisite:** MMP1 book (multivariate MP decomposition). SMP and MMP3 are rejected.

Fits equivalent current dipoles (ECDs) to Multivariate Matching Pursuit macroatoms, localising the brain sources of coherent multi-channel EEG activity on a template brain (fsaverage).

Mathematical foundation. An equivalent current dipole is a point source of neural current characterised by position `(x, y, z)`, orientation `(o_x, o_y, o_z)`, and moment `q` (nAm). The scalp potential it produces at electrode `i` is:

```
V_i = L_i(x, y, z) * q * (o_x, o_y, o_z)
```

where `L_i` is the lead-field vector from the BEM forward model.

Fitting procedure.

1. Read macroatoms from the MMP book. Each macroatom groups all channels for one MP iteration, sharing time, frequency, and scale parameters while having per-channel amplitude (MMP1 algorithm version).
2. Determine amplitude signs via circular mean of phases across channels.
3. Construct an MNE `EvokedArray` from the macroatom's spatial pattern.
4. Fit the dipole using `mne.fit_dipole()` with the BEM model.
5. Record: position (x, y, z), orientation, goodness-of-fit (% variance explained), amplitude.
6. Optionally compute distance to nearest cortical surface voxel (requires nibabel + fsaverage source space).

   First-run network requirement (fsaverage). The template head model (`fsaverage` BEM + head↔MRI transform + cortical/head surfaces) is **not bundled** in the desktop builds—the managed **curated subset** (~291 MB) is downloaded once the first time you fit dipoles or run source estimation, into `~/.obci/ide4eeg/fsaverage/`. So the first such run needs an internet connection; afterwards it runs fully offline from that cache. If you are offline on first use, it fails with a clear message telling you where to

put a manually-downloaded copy: place the `fsaverage` subject folder at `~/.obci/ide4eeg/fsaverage/`. See **Template mode (fsaverage)** below for the full resolution order and the exact file list.

Atom selection (not classification): `freq_min`/`freq_max`, `scale_min`/`scale_max`, and `amp_min`/`amp_max` filter *which* MP atoms get a dipole fit (e.g. the spindle band 11–16 Hz, scale ≥ 0.5 s as in the `dipole_spindles` example). Each gate compares against the atom's **dominant channel** (frequency / scale / peak-to-peak amplitude) and `0` disables that bound. There is no automatic structure labelling—the output CSV carries each atom's `frequency_Hz` and `scale_s`, and you classify post hoc.

Remove atoms in artifact-marked spans (panel checkbox, on by default). When the EEG-artifact detectors have marked the signal, a macroatom whose time-center falls inside the **conditioned whole-montage artifact band** is omitted before fitting, so contaminated activity does not localise to a spurious dipole. The band is the *same* product every other consumer reads—the gray band the eye view shows, the continuous-mode PSD omission, and the segment drop (§3.12)—so "atom-center in the gray band → dropped" holds by construction. **Why whole-montage:** a MMP macroatom is channel-uniform—one time and scale shared across the whole topography—so an artifact on *any* channel at that instant corrupts the multi-channel fit; the union of every channel's marked spans is the correct scope. (EEG profiles, §4.2, read the same whole-montage band today.) It is a no-op when no artifacts were marked (safe to leave on). The Run log reports `omitting N/M macroatoms in artifact-marked (conditioned) time`. If the MP book was decomposed at a different sample rate than the current signal the two frames cannot be aligned, so rejection is skipped with a warning rather than dropping the wrong atoms.

Parameter	Default	Description
`ref_channel`	`"average"`	Reference channel for dipole fitting.
`montage`	`"standard_1005"`	Electrode montage name (MNE-compatible). `standard_1005` covers most 10-20 and 10-10 systems.
`ignored_channels`	`""`	Comma-separated channel names to exclude from the dipole fit (e.g. known-bad electrodes). Empty = use all EEG channels.
`max_iterations`	`0`	Maximum number of dipoles to fit. 0 = all atoms.
`min_gof`	`0`	Minimum goodness-of-fit (%). 0 = no constraint.
`cortical_distance`	`false`	Compute cortical distances (requires nibabel).
`freq_min/freq_max`	`0`	Atom frequency filter (Hz). 0 = no constraint.
`scale_min/scale_max`	`0`	Atom duration filter (s). 0 = no constraint.
`amp_min/amp_max`	`0`	Atom amplitude filter (dominant-channel peak-to-peak µV). 0 = no constraint. Filters *which* atoms are fitted, like `freq`/`scale`.
`reject_artifacts`	`true`	Omit macroatoms whose time-center falls in the conditioned whole-montage artifact band (§3.12). See below.

Output files (in `<output>/analysis/dipole_analysis/`):

- `<base>___dipoles.csv`—per-dipole parameters: position (HEAD metres + MNI mm), orientation, GOF, amplitude, frequency, scale, cortical distance. ConnectiVIS reads this directly for 3D rendering—the `mni_x/y/z_mm` columns place dipoles on the cortex PLY, and `amplitude_nAm` / `gof_pct` / `frequency_Hz` drive size/colour modes.
- `<base>___electrodes.dat`—electrode positions in MNI mm for ConnectiVIS.
- `<base>___dipole_brain_*.png`—MNE 2D scatter plots (axial / coronal / sagittal) coloured by amplitude, frequency, GOF.
- `<base>___dipole_gof.png`—goodness-of-fit histogram.

Template mode (fsaverage—default). **fsaverage** is FreeSurfer's average brain model—the default reference brain when a subject's own MRI is not available. It backs dipole fitting, source estimation (MNE/dSPM/sLORETA/LCMV/MxNE), and the ConnectiVIS 3D views.

IDE4EEG provisions it as a managed tool, resolved from one fixed location and never from the config—exactly like Svarog and empi. The first time a run needs it, IDE4EEG resolves fsaverage in this order: (1) the managed

curated subset in `~/.obci/ide4eeg/fsaverage/` (~291 MB—only the files the analyses read); (2) as a last resort, MNE's full ~0.7 GB OSF download into `~/mne_data/`. When nothing is present, it auto-downloads the curated subset from GitLab (the GUI asks once before a run that needs it; you can also pre-install it in **Config → Tool Paths → Brain geometry (3D)**). Progress shows in the Run log.

What the curated subset contains. The full FreeSurfer `fsaverage` subject is ~761 MB; IDE4EEG only reads a fraction of it, so the managed package ships exactly the files below (~**291 MB**, plus the bundled `LICENSE-FreeSurfer.txt` + `ATTRIBUTION.txt`). Everything else in the full subject—extra BEM resolutions, the inflated/curv surfaces, additional atlases and source spaces—is omitted.

File	Size	Used by
`bem/fsaverage-5120-5120-5120-bem-sol.fif`	226 MB	BEM forward **solution**—dipole fitting + all source estimation (the bulk of the package)
`bem/fsaverage-trans.fif`	4 KB	head↔MRI transform—fitting, MNI mapping, 3D placement
`bem/fsaverage-5120-5120-5120-bem.fif`	364 KB	BEM surfaces
`bem/fsaverage-inner_skull-bem.fif`	484 KB	inner-skull BEM
`bem/{brain,inner_skull,outer_skull,outer_skin}.surf`	4 × 364 KB	head/brain **outlines** for `plot_dipole_locations`
`surf/{lh,rh}.pial`	2 × 5.6 MB	cortical surface—ConnectiVIS brain PLY
`surf/{lh,rh}.white`	2 × 5.6 MB	white-matter surface—3D outlines
`surf/{lh,rh}.sphere`	2 × 5.6 MB	source-space subsampling (`setup_source_space`)
`surf/lh.seghead`	5.3 MB	scalp surface—head PLY
`label/{lh,rh}.aparc.annot`	2 × 1.3 MB	Desikan–Killiany atlas colours
`mri/` (whole directory)	~21 MB	`orig.mgz` + `transforms/talairach.xfm` (MNI mapping via `read_talxfm`) + the a2005s aseg (`cortical_distance`)—shipped whole to avoid under-curating MNE's internal reads

If the OSF last-resort path is ever reached and fails with a `429 Too Many Requests`, that mirror is rate-limiting (your network is fine)—wait a few minutes and retry. To install fsaverage by hand—to work offline, or to use your own copy—place the `fsaverage` subject directory at `~/.obci/ide4eeg/fsaverage/` (the folder named `fsaverage`, containing `bem/`, `surf/`, `label/`—specifically `bem/fsaverage-5120-5120-5120-bem-sol.fif` and `bem/fsaverage-trans.fif`); note this is the subject folder itself, not FreeSurfer's parent `SUBJECTS_DIR`.

The brain cortex PLY (with Desikan-Killiany atlas colours) is cached once in `~/.obci/connectivis/fsaverage_brain_aparc.ply` (327k vertices, 13 MB) and shared across all runs.

ConnectiVIS renders the brain PLY inside its built-in head model (a stylised mesh with nose, ears, neck). The head is cosmetic—not registered to MNI space, so electrodes may hover slightly above or sink into the visible scalp. The electrode and dipole positions themselves are geometrically correct in MNI coordinates.

Limitation. fsaverage has no individual facial features (they average out to a smooth surface), so ConnectiVIS uses its built-in stylised head. Dipole fitting and source estimation always run on this template brain; fitting to a subject's own FreeSurfer reconstruction is not currently supported.

Example (MP-based). See `examples/dipole_spindles/`—methodology from Durka et al. (2024) *Sensors* 24(3):842 (DOI). 24-channel sleep EEG, MMP1 decomposition (200 iterations), dipole fit with spindle selection

(11–16 Hz, scale ≥ 0.5 s). A separate, non-standard approach using multivariate Matching Pursuit. Uses fsaverage (template mode).

References:

Durka PJ, Dovgiało M, Duszyk-Bogorodzka A, Biegański P (2024) "Two-Stage Atomic Decomposition of Multichannel EEG and the Previously Undetectable Sleep Spindles." *Sensors* 24(3):842. DOI: 10.3390/s24030842

Kuś R, Różański PT, Durka PJ (2013) "Multivariate matching pursuit in optimal Gabor dictionaries." *BioMed Eng OnLine* 12:94.

4.2 EEG profiles

GUI panel: EEG Profiles—checkable · ▶👁 run-and-view · **?** help · ⚠ **needs MP book** until one is on disk
Config section: `[eeg_profiles]` **Prerequisite:** MP book (decomposition)

Detects and counts EEG graphoelements—stereotyped waveforms such as sleep spindles, alpha waves, delta waves, K-complexes—by filtering MP atoms on their Frequency, Amplitude, Scale, and Phase (FASP) parameters.

Use this for continuous (rest-mode) signals. EEG profiles are designed around a continuous time axis—graphoelement counts are binned over absolute time. The GUI flags this when you pair Profiles with event-locked mode.

Structures filtering by parameters. Each atom from the MP book is characterised by frequency `f`, amplitude `A`, scale `s`, phase `phi`. A structure template defines acceptable ranges for each parameter. An atom matches if all criteria are satisfied:

```
match = (freq_min <= f <= freq_max) AND
        (amp_min <= |A| <= amp_max) AND
        (scale_min <= s <= scale_max) AND
        (phase_min <= phi <= phase_max)
```

where 0 in any field means "no constraint" (always passes).

Preset structure templates.

Structure	Frequency [Hz]	Amplitude p2p [µV]	Scale [s]	Phase
Sleep spindles	11–16	≥ 12	0.5–2.5	—
SWA	0.5–2	≥ 75	0.5–6	—
Alpha waves	8–12	≥ 5	—	—
Delta waves	0.5–4	≥ 75	0.5–6	—
Beta waves	15–25	≥ 5	0–0.5	—
Theta waves	4–8	≥ 15	≥ 1.0	—

SWA (slow-wave activity) is the AASM N3 marker—narrower (0.5–2 Hz) than Delta waves. **Sleep spindles** and **SWA** are the staple sleep-staging structures, so they get the two clearest, maximally-contrasting lane colours (vivid **red** spindles vs strong **blue** SWA). On the **SWA percentage** panel a dashed **N3 ≥ 20 %** reference line is drawn (an epoch scores N3 when SWA fills ≥ 20 % of it); that line appears only on SWA, not on Delta waves (whose 0.5–4 Hz band is broader than the AASM SWA band).

Time-evolution metrics. Detections are binned into **pages**—intervals equal to the MP segment length (read from book metadata, usually 20 or 30 s). Four display modes:

Mode	Metric	Description
`count`	Detections per page	Total number of matching atoms per page.
`percentage`	union of per-atom spans `[t0±scale_s/2]`, clipped per page ÷ interval × 100	Fraction of the page occupied by the structure (%); overlapping atoms count once, bounded ≤100%.

Mode	Metric	Description
`p2p`	Per-occurrence markers	One stem at each detection time, height = peak-to-peak amplitude [µV].
`power`	`Σ energy / epoch`	Mean power of this structure's atoms over each page (e.g. all delta waves in the delta panel): the page's summed atom energy ($\int a^2(t)dt$, µV²·s—Svarog C-BOOK-1) divided by the page length → **µV²**. For Delta this is the **slow-wave-activity (SWA) per epoch** curve, its most useful application. (Binned per page; ÷epoch makes 20 s and 30 s pages comparable.)

Bin width is always equal to the MP segment length. It is not user-configurable—the legacy `interval_sec` config key is ignored if present.

Open in Svarog. The EEG Profiles panel header carries a button that opens Svarog with the cleaned signal, the MP book synchronised below it (the book rendering *is* the profile view), and the detected structures overlaid from the per-channel `-tag.txt` mark file—one synchronised view, so you can scrub the recording and see detected structures inline.

Outputs (in `eeg_profiles/`):

- Multi-panel time-evolution chart (PNG) with one row per structure. When the recording is ≥ 1 hour the time axis is rendered in hours; below that, in seconds.
- CSV detection table (name, iteration, t0_s, scale_s, f_Hz, amplitude_uV, energy_uV2s, phase). `amplitude_uV` is peak-to-peak µV; `energy_uV2s` is the atom energy $\int g^2 dt$ in µV²·s (the Power profile is this divided by the bin interval).
- Svarog mark overlay `<file>_profiles_<channel>-tag.txt` (Format A—MNE-Python's channel-scoped annotation `.txt`). One annotation per detected structure, labelled by name with `f/amp/energy` metadata.
- **Optional** (tick *Histograms* in the panel): per-structure histograms of atom iteration, p2p amplitude, and frequency—`<file>_profiles_<channel>_histograms.png`.

Remove atoms in artifact-marked spans (panel checkbox, on by default). When the EEG Artifacts step has marked the signal, MP atoms whose time-center falls inside a `BAD_*` span (the channel-scoped artifact carrier, taken whole-montage) are dropped before binning, so every output—chart, CSV, `-tag.txt` overlay, histograms—excludes artifact-contaminated atoms. It is a no-op when no artifacts were marked, so leaving it on is harmless.

A shown band is an applied band. When the profile view draws a gray artifact band over a profile, that profile was filtered against exactly that band—IDE4EEG never draws a band it didn't apply, so a span shown in gray is always one whose atoms were dropped from the counts, never shown-but-counted.

> **Reference:** Malinowska U, Chatelle C, Bruno MA, Noirhomme Q, Laureys S, Durka PJ (2013) "Electroencephalographic profiles for differentiation of disorders of consciousness." *BioMedical Engineering OnLine* 12:109. DOI: 10.1186/1475-925X-12-109

4.3 Connectivity

GUI panel: Connectivity—checkable · ▶ run-and-view · **?** help **Config section:** `[connectivity]`

Estimates directed and undirected information flow between EEG channels. Built on ConnectiviPy.

Multivariate Autoregressive (MVAR) model. The k-channel EEG signal is modelled as:

```
X(t) = A_1 * X(t-1) + A_2 * X(t-2) + ... + A_p * X(t-p) + E(t)
```

where `A_m` are (k × k) coefficient matrices, `p` is the model order, and `E(t)` is Gaussian noise with covariance `V`.

Model order is selected via Akaike Information Criterion: `AIC(p) = N * log(det(V(p))) + 2 * p * k²`, `p_optimal = argmin AIC(p)`.

MVAR fitting methods.

Method	Key	Description
Yule-Walker	`yw`	Solves the matrix Yule-Walker equations. Fast, default.
Nuttall-Strand	`ns`	Forward-backward lattice. Better for short data.
Vieira-Morf	`vm`	Harmonic-mean normalised lattice. Most robust.

Spectral decomposition. From the fitted MVAR coefficients:

```
A(f) = I - sum_{m=1}^{p} A_m * exp(-j*2*pi*f*m/fs)
H(f) = A(f)^(-1)          -- transfer function
S(f) = H(f) * V * H(f)^H  -- cross-spectral density
```

Connectivity measures.

Method	Type	Formula	Interpretation
DTF	AR	`DTF(f)_{i->j} = abs(H_ij) / sqrt(sum_k abs(H_ik)^2)`	Directed transfer from j to i, normalised by total outflow.
gDTF	AR	DTF weighted by noise covariance	Accounts for differing noise levels across channels.
ffDTF	AR	`abs(H_ij) / sqrt(sum_f sum_k abs(H_ik(f))^2)`	Full-frequency normalisation.
dDTF	AR	`DTF * Partial_Coherence`	Direct DTF: removes indirect (mediated) paths.
iDTF	AR	Instantaneous DTF	Captures contemporaneous interactions.
PDC	AR	`PDC(f)_{i->j} = abs(A_ij) / sqrt(sum_k abs(A_kj)^2)`	Partial directed coherence: direct influence at the input side.
gPDC	AR	PDC weighted by inverse noise covariance	Generalised PDC.
iPDC	AR	Instantaneous PDC	Zero-lag PDC variant.
PCoh	AR	Partial coherence	Undirected, controls for volume conduction.
Coh	Signal	Standard coherency from FFT	Undirected frequency-domain correlation.
PSI	Signal	Phase Slope Index	Directed, based on the slope of the phase spectrum.
GCI	AR	Granger Causality Index	Directed, based on prediction error reduction. Fits an MVAR model, so it honours `mvar_method` / `mvar_order`.
AEC	Signal	Amplitude Envelope Correlation	Undirected, correlation of Hilbert envelopes.

Short-time connectivity is optionally computed in a sliding window to track dynamics over time.

Parameter	Default	Description
`methods`	`["dtf", "coh"]`	List of connectivity measures.
`mvar_method`	`"yw"`	MVAR fitting method.
`mvar_order`	`0`	AR model order. 0 = automatic via Akaike.
`resolution`	`100`	Frequency resolution for spectral estimation.
`channels`	`"all"`	Channels to include. `"all"` (or absent) = **all EEG channels only** (STIM/EOG/reference excluded); pass a subset list to restrict.

Parameter	Default	Description
`short_time`	`false`	Enable sliding-window connectivity.
`st_window`	`0`	Short-time window length (s). 0 = auto.
`st_overlap`	`0`	Short-time window overlap (s).
`significance`	`false`	Compute bootstrap significance thresholds.
`sig_reps`	`100`	Bootstrap repetitions.
`sig_alpha`	`0.05`	Significance level.

Parallelism. The bootstrap / surrogate / short-time inner loops in `connectivity/conn.py` are dispatched via `joblib.Parallel`, inheriting `n_jobs` from `[parallelism]`. MVAR-based methods (DTF, PDC, iDTF, iPDC, gDTF, gPDC, dDTF, ffDTF) typically see 3–5× speedup on 8 cores when significance or short-time is on. Coherency/PSI/GCI per-rep work is small enough that joblib spawn overhead competes—those paths show flat or marginally-slower wall time for small inputs but break even on longer recordings.

AEC notes. AEC bandpass-filters the signal in five default bands (theta [6, 7], alpha [8, 13], beta [15, 25], low-gamma [30, 45], high-gamma [55, 70] Hz). Bands whose upper edge would exceed the Nyquist frequency are skipped with a one-shot warning rather than aborting (e.g. high-gamma drops out at fs ≤ 140 Hz, including standard 128 Hz BrainTech).

ConnectiVIS .dat export. For every successfully-computed method, the connectivity step writes one `.dat` file per frequency band so the user can scrub bands × methods in 3D.

Band	Range (Hz)
`fullband`	0 → Nyquist
`delta`	1–4
`theta`	6–7
`alpha`	8–13
`beta`	15–25
`low-gamma`	30–45
`high-gamma`	55–70

Bands are defined locally in `connectivity_analysis.py::_CONNECTIVIS_BANDS` (separate from the AEC `FQ_BANDS` registry—adding `delta` does not affect AEC). `fullband` averages all bins from DC to Nyquist. Bands whose upper bound exceeds Nyquist are skipped with an info-level log line.

Filenames follow `<base>___<tag>_<data_label>_<method>_<band>.dat`. A run with DTF + PDC selected produces 14 files (2 methods × 7 bands). Per-file headers include `;title` and `;band` so each file self-describes. Significance and short-time results are intentionally not exported to `.dat`—their shapes (CI matrices, k×k×T tensors) need a different scheme. They remain available as PNG plots.

Output. Per-method directional plots, channel × channel heatmaps, PSD plots, frequency-resolved connectivity, optional short-time spectrograms, and the `.dat` files for ConnectiVIS.

Signed methods in the figures. PSI is signed—the sign is the direction of flow—so its matrix and frequency figures preserve it: a diverging colour scale centred on zero, and a frequency axis built from PSI's own sub-band centres rather than a linear grid. The unsigned methods (DTF, PDC, coherence, …) are shown as magnitudes, as before.

References:

Kamiński M, Blinowska KJ (1991) "A new method of the description of the information flow in the brain structures." *Biol Cybern* 65:203–210.

Baccalá LA, Sameshima K (2001) "Partial directed coherence." *Biol Cybern* 84:463–474.

Blinowska KJ, Kuś R, Kamiński M (2004) "Granger causality and information flow in multivariate processes." *Phys Rev E* 70:050902.

MNE-wrapped analyses

Below the *Analyses from MNE:* separator, the Analysis tab exposes the MNE-Python catalog as **six collapsible category panels**—ERP, Spectra, Time-frequency, Spatial, Comparison, and Inverse solutions. Each panel has its own enable checkbox, a category-level **?** help button, an inline ▶ Run button, and a list of individual entries with per-entry [?] buttons that link straight to the MNE documentation. A few entries have inline parameter fields (frequency range, number of time points, …); the rest run with MNE's defaults.

Config key: `mne_catalog` (list of analysis IDs). Outputs land in the `MNE/` subdirectory of `IDE4EEG_OUT_*`.

Per-category channel selection

Four of the six category panels—**ERP**, **Spectra**, **Time-frequency**, **Comparison**—each carry a single **Channels** checkbox panel at the top of the section (same widget used by Connectivity and the MMP→dipole analysis). All *channel-aware* entries within that category share the panel's selection at run time. ("Channel-aware" = an entry that plots or analyses per-channel data—a butterfly trace, a PSD overlay—as opposed to a whole-scalp topography.) The remaining categories (Spatial, Inverse solutions) have no panel—channel selection would distort topographic maps or break the inverse problem.

The 13 channel-aware entries are: `erp_quality`, `erp_butterfly`, `erp_image`, `gfp`, `sme`, `cluster_permutation`, `evoked_image`, `compare_per_channel` (ERP) · `psd_multitaper`, `psd_welch` (Spectra) · `tfr_morlet`, `tfr_multitaper` (Time-frequency) · `compare_evokeds` (Comparison). `erp_joint` is intentionally excluded—`plot_joint` needs the full montage to draw the topomap inserts at GFP peaks; a small subset crashes the underlying MNE call.

Default = all EEG channels. Untick channels to restrict the whole category's analyses to a region of interest (e.g. uncheck everything except motor-cortex electrodes in the Time-frequency panel while ERP butterfly elsewhere still shows whole-scalp). Filter buttons on the panel (All / None / EEG only) speed up bulk picks.

> **TOML back-compat.** Old per-entry `mne_params.<aid>.channels = "Fz, Cz"` configs continue to work at runtime (the resolver accepts both list and comma-string forms). Save the config from the GUI to migrate to the per-category panel.

Validation: unknown channel names log a warning and are dropped from the subset; if NONE of the requested names exist in the data, that analysis raises an error (visible in the Run-tab log) and the rest of the catalog continues.

TOML keys: - **GUI persistence:** `mne_channels.<Category>`—`list[str]` (subset) or `"all"`. Per-category panel state, restored after the signal loads. - **Runtime (auto-injected):** `mne_params.<analysis_id>.channels`—`list[str]`. Set by the GUI from the per-category panel selection right before pipeline launch (omitted when the panel is "all"); headless callers may still write either form.

4.4 ERP

Panel: ERP (MNE) · ▶ run · **?** help · ⚠ **needs event marks** · *available for* event-locked epochs (some entries need 2+ event types)

The ERP—Event-Related Potential—is the trial-averaged waveform per condition:

```
ERP(t) = (1/N) · Σ_i epoch_i(t)
```

Phase-locked neural responses sum constructively while non-phase-locked noise cancels out.

ID	Name	Requires	Parameters
`erp_quality`	ERP quality stats (customized)	2+ tags	`win_start`, `win_stop`, `n_permutations`
`erp_butterfly`	ERP butterfly	—	—
`erp_image`	ERP image (per channel)	—	—
`erp_joint`	ERP joint (topos at peaks)	montage	—
`erp_topomap`	ERP topomap series	montage	`n_times`

ID	Name	Requires	Parameters
`gfp`	Global field power	—	—
`sme`	SME (data quality)	—	`win_start`, `win_stop`
`cluster_permutation`	Cluster permutation test	2+ tags	`step_down_p`, `n_permutations`
`evoked_image`	Evoked image (channels x time)	—	—
`compare_per_channel`	Compare evokeds (per channel)	2+ tags	—

ERP quality stats (customized). IDE4EEG's own per-channel report for a two-condition ERP contrast (e.g. target vs non-target)—a *customized* analysis (flagged in orange in the GUI), not a stock MNE function: it composes MNE/SciPy primitives (`mne.stats.permutation_cluster_test`, `scipy.stats.mannwhitneyu`) into one row per (condition-pair, channel). For each channel over the measurement window it reports: the **effect** (`eff`—first condition (`t`) minus second (`nt`) mean amplitude); the single-trial separability **AUC** (`auc`/`p_mw`; the normalized Mann–Whitney U); and a window-free per-channel temporal **cluster mass** (`cl_mass`—$\Sigma|F|$ over the significant permutation cluster) with its significant window (`cl_from`/`cl_to`). This cluster test runs on each channel **independently** (its trials × time)—unlike the `cluster_permutation` entry's **spatio-temporal** test, which pools channels via spatial adjacency and can therefore mark a whole band of channels significant at once. So per-channel significance differs between the two: a channel that is non-significant here (blank window, `cl_mass = 0`) may still fall inside a montage-wide spatio-temporal cluster. The window-free cluster mass and the single-trial AUC are complements to the windowed effect, because reading a result off the same amplitude one is measuring is circular analysis (Kriegeskorte et al. 2009); a channel can thus have a small or negative *windowed* effect yet a strongly significant *window-free* cluster elsewhere in the epoch. For a per-condition measurement-precision (data-quality) readout, use the **SME (data quality)** analysis above. Restrict the channels via the shared ERP channel panel; the defaults measure a standard P300 window. Writes `<file>___erp_quality.csv`; requires ≥2 event types.

Parameter	Default	Description
`win_start`	`0.25`	Start of the P300 measurement window (s); standard P300 range, adjust per subject.
`win_stop`	`0.5`	End of the measurement window (s); `stop <= start` means the end of the epoch.
`n_permutations`	`1024`	Permutations for the per-channel temporal cluster test (the cluster mass).

References: *ERP quality stats*—Kriegeskorte N, Simmons WK, Bellgowan PSF, Baker CI (2009) "Circular analysis in systems neuroscience: the dangers of double dipping." *Nat Neurosci* 12(5):535–540.—*SME (data quality)*—Luck SJ, Stewart AX, Simmons AM, Rhemtulla M (2021) "Standardized measurement error: A universal metric of data quality for averaged event-related potentials." *Psychophysiology* 58(6):e13793.

Spatio-temporal cluster permutation test (Maris & Oostenveld 2007). Non-parametric method controlling the family-wise error rate when testing for differences across many time points and channels, without overly conservative corrections like Bonferroni. Per time point and channel, compute a t-statistic comparing two conditions; threshold to identify above-significance samples; group adjacent above-threshold samples (in time and space) into clusters; sum statistics within each cluster; randomly reassign condition labels and repeat (default `n_permutations = 1024`); report the fraction of permutations where the maximum cluster statistic exceeds the observed cluster statistic. Wraps `mne.stats.spatio_temporal_cluster_test` with channel adjacency from the electrode montage.

Parameter	Default	Description
`step_down_p`	`0.05`	P-value for step-down-in-jumps test.
`n_permutations`	`1024`	Number of permutations (higher = more precise).

Reference: Maris E, Oostenveld R (2007) "Nonparametric statistical testing of EEG- and MEG-data." *J Neurosci Methods* 164(1):177–190.

4.5 Spectra

Panel: Spectra (MNE) · ▶ run · **?** help · *available for* rest or epochs

ID	Name	Requires	Parameters
`psd_topomap`	PSD topomap	montage	`bands`, `scale` (`"dB"`/`"power"`), `normalize`
`psd_multitaper`	PSD multitaper	—	`fmin`, `fmax`, `scale` (`"dB"`/`"power"`/`"amplitude"`), `bandwidth`, `tmin`, `tmax`, `xscale`, `average`, `ci`
`psd_welch`	PSD Welch	—	`fmin`, `fmax`, `scale` (same three choices), `n_per_seg`, `n_overlap`, `average`, `xscale`
`psd_topo`	PSD per-channel (sensor layout)	montage	`fmin`, `fmax`, `scale` (`"dB"`/`"power"`)

`psd_topomap` covers the previous `psd_bands_topomap` use case via its `bands` parameter (the two were merged into one entry). Line-plot entries (`psd_multitaper`, `psd_welch`) expose three `scale` choices (`dB` / `power` / `amplitude`); topographic entries only the first two (no `amplitude`). All four PSD entries also carry an advanced `save_csv` parameter (bool, default off)—tick it to export the computed spectrum to a `.csv` alongside the plot.

Rest mode—continuous estimation that omits artifacts. In **rest / continuous** mode all four PSD entries estimate on the *continuous* preprocessed signal rather than on the cut windows, and omit artifact-marked spans via MNE's `reject_by_annotation`—so a sub-window blink or muscle burst is excluded at full granularity (with the same `min_mark_ms` / `dilate_ms` conditioning the [Mark detected artifacts] step applies, §3.12) instead of contaminating a whole window. This is controlled by `[segmentation].reject_by_annotation` (default `true`; set `false` to estimate on the raw continuous signal with no omission). Time-frequency (TFR) entries stay **windowed**—they need an intact epoch time axis and cannot drop sub-spans (§4.6). In event-locked mode every spectral entry is computed on the epochs as before.

4.6 Time-frequency

Panel: Time-frequency (MNE) · ▶ run · **?** help · *available for* rest or event-locked epochs (ERD/S Topomap is event-locked-only)

All three TFR entries call `Epochs.compute_tfr(method=...)` (MNE ≥ 1.4 API).

ID	Name	Requires	Parameters
`tfr_morlet`	TFR Morlet wavelet	—	`freq_min`, `freq_max` (0 = Nyquist), `n_cycles_mode` (`"adaptive"`/`"constant"`), `n_cycles`
`tfr_multitaper`	TFR multitaper	—	`freq_min`, `freq_max`, `n_cycles_mode`, `n_cycles`, `time_bandwidth` (default 4.0, advanced)
`erds_topomap`	ERD/S topomap	montage	`freq_min`, `freq_max`, `n_cycles_mode`, `n_cycles`

`n_cycles_mode` picks the time-frequency-resolution trade-off: - `"adaptive"` (default): the number of cycles grows with frequency (`n_cycles_i = freqs_i / value`), so low frequencies use a wider—longer—time window (MNE-tutorial convention). Default `value = 2` gives 5 cycles at 10 Hz, 25 cycles at 50 Hz. - `"constant"`: `n_cycles` cycles at every frequency (Tallon-Baudry convention; commonly 7).

Wavelet length must fit the epoch. MNE's Morlet kernel is truncated to $\pm 5\sigma$ where σ = `n_cycles / (2πf)` (σ is the width of the wavelet's Gaussian envelope; $\pm 5\sigma$ captures essentially all its energy), giving an effective duration of $\approx$ `1.59·n_cycles/f` seconds. At fmin this is the longest kernel in the bank—and the formula depends on which `n_cycles_mode` is active:

- **Adaptive mode (default)**: `n_cycles[i] = freqs[i] / value`, so the kernel at fmin is ≈ `1.59 / value` seconds—**independent of fmin**. With default `value=2`, that's ~0.8 s, comfortably inside typical event-locked epochs. This warning rarely fires here.
- **Constant mode**: `n_cycles = value` at every frequency, so the kernel at fmin is ≈ `1.59·value / fmin` seconds. With `value=7` (Tallon-Baudry convention) and `fmin=1 Hz` that's ~11 s—needs an 11+ second epoch. This mode is where the wavelet-too-long warning typically bites.

When the kernel exceeds the epoch length, the low-frequency rows of the TFR are essentially noise: MNE itself warns ("at least one of the wavelets is longer than the signal"). IDE4EEG pre-empts MNE's terse message with an actionable line naming the entry, the kernel duration, and the epoch length—visible on the Run-tab log. Fix any of: raise `freq_min`, lower `n_cycles`, switch to adaptive mode, or use longer epochs. Multitaper uses DPSS windows with comparable nominal duration; the same check applies.

ERD/S Topomap behaviour: the catalog entry first computes a Morlet TFR per condition, then calls `power.apply_baseline(baseline=(None, 0), mode="percent")`. The baseline window is fixed to the **entire pre-event interval** (epoch start → event onset at t=0 s, relative time). The colorbar reads in %: negative values = ERD (event-related desynchronization, i.e. power decrease), positive values = ERS (synchronization, power increase). The cmap is diverging (`RdBu_r`) centred at 0.

4.7 Spatial

Panel: Spatial (MNE) · ▶ run · **?** help · *available for* rest or epochs (Evoked topomap series is event-locked-only)

ID	Name	Requires	Parameters
`evoked_topomap_anim`	Evoked topomap series	montage	`n_times`
`channel_locations`	Channel locations	montage	—

4.8 Comparison

Panel: Comparison (MNE) · ▶ run · **?** help · ⚠ **needs event marks** · *available for* event-locked epochs

ID	Name	Requires	Parameters
`compare_evokeds`	Compare evokeds	2+ tags	`ci`
`drop_log`	Drop log	—	—

4.9 Inverse solutions

Panel: Inverse solutions (MNE) · ▶ run · **?** help · ⚠ **needs event marks** · *available for* event-locked epochs

Four source-estimation entries that share `[dipole_fitting]` paths / BEM / trans configuration but expose their own per-entry parameters. The output of every entry is also wrapped into ConnectiVIS scene companion files (brain PLY, head PLY, electrodes DAT) so the **3D view** button on the Output tab works identically across them.

ERP dipole fit (`erp_dipole_fit`)

Standard MNE dipole-fit tutorial: average epochs per condition → compute noise covariance from baseline → `mne.fit_dipole(evoked, cov, bem, trans)` at each time point. No MP decomposition needed—the classical approach.

Parameter	Default	Description
`time_min`	`0.0`	Start of time window to fit (s). 0 = event onset.
`time_max`	`0.0`	End of time window. 0 = full epoch.
`min_gof`	`50`	Minimum goodness-of-fit (%).
`min_dist` *(advanced)*	`5.0`	Minimum distance (mm) the dipole keeps from the inner skull.

Parameter	Default	Description
`pos` *(advanced)*	`""`	Optional `"x,y,z"` in head-coordinate millimetres to fix location (orientation + amplitude only).

`n_jobs` is read from `[parallelism]` and passed straight to `mne.fit_dipole`. Outputs: cross-section plots, `___erp_dipoles.csv`, and ConnectiVIS companion files. Reference: MNE dipole fit tutorial.

MNE / dSPM / sLORETA / eLORETA inverse (`mne_inverse`)

Minimum-norm distributed-source family. Builds a forward solution + inverse operator from the epochs and applies it to the per-condition Evoked. Output is a cortical activation per vertex (~2.5k–8k vertices × time, depending on `spacing`), saved as a pair of `*-lh.stc` + `*-rh.stc` FIF files (one per cortical hemisphere—MNE's native source-estimate format; reload with `mne.read_source_estimate(stem)`) plus a static PNG screenshot per condition rendered via PyVista.

Parameter	Default	Description
`method`	`"dSPM"`	`"MNE"` / `"dSPM"` / `"sLORETA"` / `"eLORETA"`.
`snr`	`3.0`	Signal-to-noise ratio ($\lambda^2 = 1/\mathrm{SNR}^2$).
`loose` *(advanced)*	`0.2`	Source orientation constraint (0 = fixed, 1 = free).
`depth` *(advanced)*	`0.8`	Depth-weighting exponent.
`spacing` *(advanced)*	`"ico4"`	Source-space subdivision.
`n_peaks` (GUI label: "Strongest sources to export")	`5`	Extract the N cortical vertices with the strongest source amplitude (per-vertex peak over time, then top-N) and export them as `___mne_inverse_<method>_dipoles.csv` + `___electrodes.dat` for ConnectiVIS 3D view. Lossy by construction (collapses ~2.5k–8k smooth vertices to N point sources); the full FIF `-lh.stc`/`-rh.stc` pair is always saved separately. `0` = skip the CSV (FIF + PNG only).

Reference: MNE inverse tutorial.

LCMV beamformer (`lcmv_beamformer`)

Linearly Constrained Minimum-Variance beamformer (`mne.beamformer.make_lcmv` + `apply_lcmv`). Same output shape as `mne_inverse`.

Parameter	Default	Description
`reg`	`0.05`	Tikhonov regularisation of the data covariance.
`pick_ori` *(advanced)*	`"max-power"`	`"max-power"` / `"normal"` / `"vector"`.
`weight_norm` *(advanced)*	`"unit-noise-gain"`	`"unit-noise-gain"` / `"nai"` / `"none"`.
`spacing` *(advanced)*	`"ico4"`	Source-space subdivision.
`n_peaks` (GUI label: "Strongest sources to export")	`5`	Same as `mne_inverse` above—top-N beamformer-output peaks → dipoles CSV + electrodes DAT for ConnectiVIS. `0` = skip the CSV.

Reference: MNE LCMV tutorial.

MxNE / iRMxNE sparse (`mxne`)

`mne.inverse_sparse.mixed_norm`. Returns a handful of focal sources (~5–20 vertices) instead of a smooth distribution—feeds ConnectiVIS automatically via `___mxne_dipoles.csv`, no `n_peaks` opt-in needed.

Parameter	Default	Description
`alpha`	`80.0`	Sparsity / regularisation (% of `α`_max). 50–90 typical for ERP.
`depth` *(advanced)*	`0.9`	Depth-weighting exponent.
`loose` *(advanced)*	`0.2`	Source orientation constraint.
`n_mxne_iter` *(advanced)*	`1`	1 = vanilla MxNE. ≥ 2 enables iterative re-weighted MxNE (iRMxNE).
`spacing` *(advanced)*	`"ico4"`	Source-space subdivision.

Reference: MNE mixed-norm example.

Choosing a source-estimation entry

Method	Output shape	ConnectiVIS path
`erp_dipole_fit`	one ECD per time sample	direct (`___erp_dipoles.csv`)
`mxne`	~5–20 sparse vertices	direct (`___mxne_dipoles.csv`)
`mne_inverse` (MNE/dSPM/sLORETA/eLORETA)	per-vertex cortical map	top-N peaks via `n_peaks` (default 5)
`lcmv_beamformer`	per-vertex cortical map	top-N peaks via `n_peaks` (default 5)

The discrete-source methods feed ConnectiVIS natively; distributed methods always write `*-lh.stc` + `*-rh.stc` FIF pairs (one per cortical hemisphere) plus a static PNG, and by default also extract the `n_peaks=5` strongest cortical vertices to `*_dipoles.csv` for ConnectiVIS (set `n_peaks=0` to opt out of the CSV—the FIF .stc files are unaffected).

> **Units of the source-estimate CSV `amplitude_nAm` column.** This depends on the inverse method. **Current-density methods**—`mne_inverse` with `method = MNE` or `eLORETA`, **MxNE**, and an unnormalised LCMV (`weight_norm = "none"`)—produce source amplitudes in A·m, written as **true nano-ampere-metres** (scaled $\times 10^9$), consistent with the dipole-fit CSVs. **Noise-normalised methods**—`dSPM`, `sLORETA`, and the **default** unit-noise-gain LCMV—produce a **dimensionless** statistic (a t-/z-like value) with no nAm meaning; that value rides the `amplitude_nAm` column **raw** so ConnectiVIS can still size/colour by it—a known label mismatch (a dedicated `stat_value` column is a possible future refinement). Each row also carries an `amplitude_unit` column naming what its `amplitude_nAm` value is—`nAm` for the current-density methods, or `dSPM` / `sLORETA` / `NAI` for the noise-normalised ones—so the CSV is self-documenting; ConnectiVIS ignores the extra column (it reads columns by name). Either way the ConnectiVIS 3D view is unaffected—it sizes by *relative* magnitude within a run.
>
> **References:** all MNE-wrapper analyses cite Gramfort A et al. (2014) "MNE software for processing MEG and EEG data." *NeuroImage* 86:446–460. Each entry's [?] button also links to the relevant MNE tutorial.

---

5. Run tab

The Run tab is where you actually execute the pipeline. It shows the live log, a per-stage checklist with progress bars, and a Stop button to cancel an in-progress run. The same code path is used for the GUI's **Run Pipeline** button, the per-analysis [**Run**] buttons on the Analysis tab, and the command-line `ide4eeg --run myconfig.toml`.

A typical session: configure the steps you want on Preprocess and Analysis, then switch to the Run tab and click **Run Pipeline**.

Run Pipeline button

Executes the complete pipeline: all checked preprocessing steps followed by all checked analyses. The current GUI state is collected into a config dict and saved as `config.toml` in the output directory before execution starts.

- Every checked analysis runs.
- A stage checklist shows progress in real time, with per-stage progress bars.
- For multi-file batches, stages reset between files and the overall bar tracks cross-file progress.
- All output files (plots, tables, `.fif` epochs) are written to the `IDE4EEG_OUT_*` directory.
- The Output tab refreshes automatically when the pipeline finishes.

A Run never opens a review. Clicking **Run Pipeline** executes the configured pipeline end to end without stopping: no bad-channel, ICA-component, epoch or gaze review window opens, so the result is always the pure function `f(input, config)` of the loaded recording and the current settings. Reviews are **on-demand**—you open one yourself from a step's 🔒/✏ decision button on the Preprocess tab, and a decision you made there is remembered for the recording and re-applied silently on every later Run (see When a manual review resets). The one exception is the opt-in guided sweep: with ↳ **stop at each user-editable step** armed on the Config tab, the button reads **"▶ Run Pipeline (with stops)"** and the run *does* pause at each enabled editable step. A batch `--run` never stops, under any setting.

Manual-decision summary bar. Just below the button row, a ✋ strip appears whenever the loaded recording carries remembered manual review decisions. It reads *"Manual review active this run —"* followed by each step and its decision (`bad channels: T7, O2`; `ICA exclusion: exclude IC0, IC3`; `epoch rejection: reject 4 epoch(s)`; `gaze not-looking: 2 not-looking interval(s)`), so everything a Run will apply on top of the automatic result is visible in one place without opening each panel. The strip is **hidden when there is nothing to show**—an empty Run tab means the run is fully automatic. **Revert all to automatic** beside it forgets every remembered decision for this recording at once; the next Run is then purely `f(input, config)`.

Input validation

IDE4EEG validates all numeric configuration parameters before running the pipeline. Invalid values are caught at two levels:

GUI (live feedback). Key numeric fields turn red as you type when the value is out of range:

- **Filter frequencies:** highpass must be < lowpass (when both are set); neither can be negative.
- **Frequency range:** F start must be < F end; neither can exceed the Nyquist frequency (half the sampling rate, accounting for resampling).
- **Epoch offsets:** start offset must be < stop offset.

When you click **Run Pipeline**, a dialog warns about invalid frequency ranges and lets you abort.

Consistency rules (pre-flight validation)

Beyond the numeric range checks above, IDE4EEG ships a registry of *consistency rules* covering things that the pipeline would otherwise discover too late (typos, stale configs reused across recordings, step-ordering advice between preprocessing steps (warnings only), dipole/EEG-profile dependencies on Matching Pursuit, etc.). Five fire-points are declared; **three currently carry rules** (`signal_load`, `field_edit`, `pre_launch`). `step_entry` is wired but has no registered rules, and `config_load` is reserved with no caller yet:

Fire-point	When it runs	Status
`config_load`	(reserved) when a TOML config is loaded	no caller yet
`signal_load`	When a recording is selected / loaded	active
`field_edit`	Live, after every edit to a rule-triggered field	active
`pre_launch`	When you click **Run Pipeline** (or `--run` in CLI)	active
`step_entry`	Just before each preprocessing step runs	wired, no rules yet

The same rule set runs in both the GUI and the CLI, so a config that passes preflight in one passes in the other.

Field-validation badges (live feedback). Some fields paint a small inline glyph the moment their value becomes inconsistent with the loaded signal or the rest of the config:

- **Red** ×—error (Run Pipeline will refuse to start)
- **Orange** ⚠—warning (Run will surface a confirm dialog)
- **Blue (i)**—information / hint

Hover the glyph for the full rule message and the rule-id (useful for filing bug reports). Badges currently appear next to:

- *ICA → EOG channels*: missing channel name from the signal
- *Dipole → Reference channel*: missing or invalid reference
- *Matching Pursuit → Algorithm*: cascade target – editing *Dipole fitting* may surface "Use MMP1 for dipole sources" here
- *Segmentation setup → Segmentation mode*: event-locked mode selected but the recording has no event markers (no STIM channel, no annotations, no .tag/.xml events)
- *Re-reference panel header*: missing channel in the loaded cap
- *Channel-selection panels* (Channels, MP channels, Connectivity channels, Ignored dipole channels, Event tags): stale selections
- *MP Book panels* (Dipole, EEG Profiles): pinned book on disk has been deleted or moved (paints on both panels simultaneously)
- *Preprocess tab top strip*: step-order advice violations – a global status line visible even while individual step panels are collapsed. These are warnings only; no ordering is ever refused

When the issue is resolved the badge disappears automatically.

Panel-header status surface. Some rules emit on a field that doesn't have its own widget (e.g. "Dipole fitting requires MMP1 to be in the pipeline"). Those messages render inline on the panel's header label, alongside the existing "needs MMP1 book" text. The header message clears automatically once you fix the underlying problem.

Pipeline invariants (Help tab). The **Pipeline invariants** section at the bottom of the **Help** tab lists every registered rule, grouped by module (seven modules: Reference, Dipole, Channels, Modes, Step order, Tools, Numeric ranges). Each entry shows the rule-id, the fire-points it runs at, and a short rationale. This catalogue is generated live inside the running application from the currently registered rules and updates automatically as rules are added.

Pre-flight dialog (clicking Run Pipeline). Any rule that emitted at `pre_launch` shows up in the dialog: errors block execution (no *Run anyway* button), warnings just confirm. The CLI runs the same rules at launch: errors raise `ValueError`, warnings log via `logging.warning` and the run continues.

"Don't show this again" per-rule dismissal. Right-click any badge to dismiss the rule that fired it. Dismissals persist across sessions in `~/.obci/ide4eeg/stage5_dismissed.json`. A dismissal only silences the inline badge and panel-header hints; the pre-launch preflight dialog still surfaces every rule, so you can't accidentally bypass a run-time error.

To restore a dismissed rule: open the **Pipeline invariants** section at the bottom of the **Help** tab. Once you have dismissed at least one rule, a *Dismissed rules* list appears there (a fresh install has none); find the rule-id in that list and click *Restore* next to it.

CLI validation. The command-line path runs the **same consistency rules** as the GUI pre-flight—it does not silently clamp values. At launch it builds an internal snapshot from your config (plus a peeked signal-info when the input file is readable), dispatches the `pre_launch` rules, and translates issues by severity:

- **error** → raises `ValueError` on the first one and aborts the run (with `-t` you get the full traceback). Fix the config and re-run.
- **warning** → logged via `logging.warning` and the run continues.

So an out-of-range `filters` band or a `trim_start ≥ trim_end` is an error that stops the batch, not a value that gets quietly rewritten. (A directory input carrying a legacy `manual_*` key is *not* an error—the key is stripped at load with a warning and the batch runs fully automatic; see batch = no manual.) The identical rules power the GUI pre-flight dialog, so a config that runs headless runs in the GUI and vice-versa.

Live log panel

Below the progress bar, a scrolling log panel shows the same colored output that goes to the terminal in CLI runs (INFO / WARNING / ERROR levels). The panel is always read-only; you can select and copy text from it normally.

A persistent copy is written to `~/.obci/ide4eeg/logs/ide4eeg.log`—the same messages, timestamped and rotating (3 files). Unlike the on-screen panel (in-memory, cleared each launch), it survives across sessions, so it is the file to attach to a bug report. A helper-app (Svarog / ConnectiVIS) launch that *fails* records its real error there and in the pop-up dialog, even with **Verbose console logging** off; turning verbose on additionally streams the helper's full live output into the log.

The log is capped at 50 000 blocks (~a few MB of text) so long pipelines emitting tens of thousands of log lines (ICLabel, empi) don't grow QTextEdit's document into the multi-GB range.

The **Auto-scroll** checkbox at the top right controls whether new lines automatically scroll into view; **Clear** wipes the panel.

Stopping a run

The **Stop** button (next to **Run Pipeline**) is enabled only while a pipeline is running. Clicking it sets a cancel flag that all long-running loops in the pipeline check cooperatively—most stages bail within a second. Two kinds of stage are slower: **MP decomposition** finishes its current empi invocation, and an **already-running MNE analysis** (a cluster-permutation test, TFR, or source estimation) runs to completion—the flag is checked *between* catalog analyses, not inside a single MNE call.

When a run is cancelled, partial outputs already on disk are kept (per-step snapshots, MP books), and the next run picks up from them. See Hash-based caching (overview) for how that starting point is chosen.

What gets saved on success or failure

When the pipeline runs (via **Run Pipeline** or per-analysis [**Run**]), output files land in `IDE4EEG_OUT_<filename>/`:

- **Always saved:** clean epochs as `-epo.fif` in `saved_signals/`; the MP atom book in `mp_decomposition/` (if MP decomposition ran); the exact config as `config.toml` for reproducibility.
- **Always saved when the step runs (not save-gated):** the EEG-artifact marks (`eeg_artifacts/<file>-tag.txt`) and the gaze marks + review JSON (`artifacts_detection/video_artifacts/`)—the marks file *is* each detector's primary output.
- **Conditionally saved (per-step Save checkboxes, default off):** intermediate signals in `saved_steps/`, ICA components in `artifacts_detection/ICA_EOG/`, filter response plots in `filters_plot/`. (The PREP-aligned bad-channel detector does not currently write diagnostic plots—the `bad_channels.save_plots` flag and the `bad_channels_detection/` directory are reserved for a planned follow-up; the Run-log per-channel diagnostic table partially fills the same role.)
- **Analysis:** plots and tables under `analysis/`—native analyses in named subfolders (`connectivity_analysis/`, `eeg_profiles/`, `dipole_analysis/`) and each MNE-catalog analysis in `analysis/MNE/<entry_id>/`.

See Chapter 6 for the full output table. On a failed file (e.g. ICLabel raises `ICAFailureError`), the pipeline aborts that file and continues with the next in a multi-file batch—partial outputs already written stay on disk.

What config is actually used?

In all cases, the GUI collects the current state of all widgets into a config dict at the moment you click Run. This means:

- The config reflects what you see in the GUI right now, not what was in the loaded TOML file.
- For per-analysis Run, the config is further modified to enable only that one analysis.
- The resulting config is saved as `config.toml` in the output directory, so you can inspect exactly what was executed.

Command-line execution

For batch / headless / scripted runs:

```
ide4eeg                          # open GUI, blank state (default)
ide4eeg myconfig.toml            # GUI with config preloaded
ide4eeg --run myconfig.toml      # batch run, no GUI
```

```
ide4eeg --run myconfig.toml -t      # batch with full Python tracebacks
ide4eeg --version                   # print version and exit
```

The batch run uses the same code path as the GUI, so results match. For programmatic use, see `ide4eeg/api.py` (`run_file()` and step wrappers). The GUI's **Export Script...** action writes a self-contained Python script reproducing the current configuration without a TOML file—useful for cluster submission or sharing reproducible analyses.

Batch processing (multiple files)

Point the **Input** at a *directory* and IDE4EEG treats every supported file in it as a batch. The GUI runs one recording at a time—clicking **Run Pipeline** on a folder is refused with a "folder input is batch-only" notice—so batches run headless via the CLI:

```
ide4eeg --run myconfig.toml          # input_path = a folder → every file
```

Each file is processed independently with the *same* config (deep-copied per file, so nothing leaks between recordings) and writes its own `IDE4EEG_OUT_<file>/`. A failed file is skipped and the batch continues.

Preparing a batch config. Tune the *automatic* pipeline on one representative recording: dial in filters, bad-channel detection, ICA selector, segmentation and artifact thresholds, checking each with the 👁 step previews. **Save Config...**, set the **Input** to the batch folder, and run it with `--run`. The saved config is pure `f(input, config)`—no manual decisions—so every file reproduces the same automatic processing. (`api.run_file()` and **Export Script...** cover scripted / cluster submission of the same config.) See Appendix F.5 for the full howto—the exported-script loop and the per-file-config route.

Batch is automatic-only—no manual review. Manual review decisions (bad channels, ICA components, dropped segments, gaze) are *per-recording*—produced by reviewing one signal—and live in the GUI **session** (RAM), not the config. They are never written to a saved config, and any legacy `manual_*` key left in a hand-edited config (`choosing_channels.manual_bad_channels`, `ICA_EOG.manual_exclude`, `segment_rejection.manual_reject_windows`, `gaze.manual_not_looking`) is **stripped at config load** and ignored—so a batch always runs fully automatic and no stale reviewed set can silently govern a run. Review each recording individually in the GUI, or—for a channel known-bad across the whole set—use `choosing_channels.dropped_channels` (a genuine config field, applied on every run).

---

6. Output tab

The Output tab is a results browser. The left side shows a tree of every file in the output directory; the right side previews the selected file (image, table, FIF metadata, MP book info, or text). The bottom row has helper-app launchers (Svarog, ConnectiVIS 3D, MNE, and the built-in MP Book Viewer).

When the pipeline finishes, the tab auto-refreshes; double-clicking a **file** dispatches it to the appropriate viewer. For a **folder**, a **single click** unveils its content (expands it in the tree) and a **double-click** opens it in your system file viewer (Finder / Explorer / file manager); collapse a folder with its disclosure triangle.

Output directory structure

Results are saved in a folder named `IDE4EEG_OUT_<input_filename>/`:

Two roots sit inside it: `preprocessing/` (signals, marks, per-step artifacts) and `analysis/` (plots and tables). The subfolders below are shown with those prefixes.

Subfolder	When written	Contents
`preprocessing/saved_signals/`	**always**	Clean epochs as `-epo.fif` (the primary preprocessing output)
`preprocessing/saved_steps/`	when a step's per-step **Save** checkbox is checked	Intermediate signal at that point as `<base>-<suffix>-raw.fif`

Subfolder	When written	Contents
`preprocessing/artifacts_detection/ICA_EOG/`	when **Save components plot and table** is checked on the ICA panel	ICA component topomaps, property plots, classification CSV, `-ica.fif`, MNE Report HTML
`preprocessing/bad_channels_detection/`	reserved (no plots written by the current PREP detector—see §3.3 Output)	*(empty)*
`preprocessing/eeg_artifacts/`	always, when the EEG-artifacts step runs	channel-scoped `<base>-tag.txt` marks (p2p / slope / hf / flat)
`preprocessing/artifacts_detection/video_artifacts/`	always, when the gaze step runs	`<base>-tag.txt` gaze marks + a review-state JSON
`preprocessing/filters_plot/`	when **Save filter plots** is checked on Filters	Filter frequency-response PNGs
`preprocessing/mp_decomposition/`	when MP decomposition runs (forced save)	MP atom book `.db` file
`analysis/connectivity_analysis/`	when a connectivity analysis runs	DTF/PDC/… matrices (`.dat`) + graphs (`.png`)
`analysis/eeg_profiles/`	when an EEG-profile (FASP) analysis runs	Profile plots + tables
`analysis/dipole_analysis/`	when a dipole analysis runs	Dipole fit plots + tables
`analysis/MNE/<entry_id>/`	when MNE-based analyses run	One subdir per catalog entry (`psd_multitaper/`, `tfr_morlet/`, `erp_dipole_fit/`, …) holding that entry's plots + CSVs

Per-step save toggles default off. The cleaned epochs in `preprocessing/saved_signals/` are *always* written—that's the primary pipeline output that all analyses depend on.

The output path is controlled by two config parameters on the **Input** tab:

- `output_path`—output root directory; defaults to the input signal folder. `IDE4EEG_OUT_<filename>` is created automatically with `preprocessing/` and `analysis/` inside.
- `overwrite_output`—if `false`, each run creates a timestamped subfolder (`YYYYMMDD_HHMMSS_<rest|epochs>`); if `true`, results are overwritten in place.

Browsing the tree

The navigation row above the tree has four buttons:

- **Current file**—show `IDE4EEG_OUT_*` results for the currently loaded signal.
- **↑ level up**—search parent directories (of the output path, input path, and last browsed folder) for additional `IDE4EEG_OUT_*` results.
- **Open...**—pick any folder; only `IDE4EEG_OUT_*` subdirectories are shown (duplicates skipped).
- **Delete selected**—remove the selected file or folder from disk.

Hidden noise (`.DS_Store`, `Thumbs.db`, `___electrodes.dat`, `___head.ply`, `___brain.ply`) is filtered from the tree—those companion files travel with their main artefact (the `dipoles.csv`) and aren't useful to browse on their own.

The preview pane (right side) auto-renders by extension:

- **Images** (`.png`, `.jpg`, `.jpeg`, `.gif`, `.bmp`)—inline display, double-click to open in the system viewer.
- **CSV**—sortable table (limited to first 500 rows).
- **Text** (`.txt`, `.log`, `.toml`, `.xml`, `.json`)—text view with monospace font.

- **FIF** (`.fif`)—metadata summary (sfreq, duration, channels, info description). Double-click the tree entry to open it in MNE.
- **MP book** (`.db`)—book metadata + **Open in Book Viewer** action (the built-in MP Book Viewer).
- **Tag** (`.tag`, `-tag.txt`)—text view of a Svarog mark/tag file (legacy BrainTech XML `.tag` or the Format-A MNE-annotation `-tag.txt`).

Opening files in helper apps

The view row below the tree has six buttons that operate on the currently selected file (left to right). Buttons that don't apply to the current selection are greyed out.

- **SVAROG**—open the selected file in Svarog. Auto-detects nearby `-tag.txt` / `.tag` and `.db` files so the launch includes the markers and atom book if present; a profiles `…_profiles_<ch>-tag.txt` opens as signal + synced book + structure overlay. Available for `.fif`, `.edf`, `.bdf`, `.raw`, `.vhdr`, `.set`, `.tag`, `-tag.txt`, `.db`.
- **3D**—open the selected `.dat` or `*dipoles.csv` in the ConnectiVIS 3D viewer.
- **all 3D**—open ALL `.dat` / `.csv` results in the current run (dipoles + connectivity + ERP-dipole) together in one ConnectiVIS scene. Greyed out when the run has no 3D-viewable results.
- **MNE**—open the selected signal file in MNE's interactive viewer (Raw, Epochs, or ICA components).
- **book**—open the selected MP book (`.db`) in the built-in Book Viewer.
- **system**—open the selected file in the OS default application, or a selected folder in the system file manager.

Double-click in the tree dispatches by extension automatically:

- Signal files (`.fif`, `.raw`, `.edf`, `.bdf`, `.vhdr`) → Svarog when a jar is installed; MNE fallback (for `.fif`) or OS default otherwise.
- `*-epo.fif` (mne.Epochs) → MNE epochs browser.
- `*-ica.fif` (mne.preprocessing.ICA) → MNE component-topomap overview.
- `.db` (empi MP book) → IDE4EEG MP Book Viewer.
- `.dat` (electrode positions / connectivity arrows) → ConnectiVIS.
- Anything else → OS default handler.

Double-click dispatch uses Svarog when a jar is installed (and current enough), falling back to MNE otherwise; the explicit view-row buttons open the named app directly.

---

7. Help tab

The Help tab is an in-app reader for this manual. The left pane is a filterable table of contents; the right pane shows the rendered Markdown with a search bar at the top.

You don't have to read the manual cover-to-cover to use IDE4EEG—every interesting widget in the GUI has a **[?]** button or a hover tooltip. The Help tab is where the long form lives, and where the [?] buttons jump to when you click their **Open in Manual** link.

Context help—the [?] buttons

Most preprocessing and analysis sections have a **[?]** button next to the panel title (and in some cases, next to individual fields). Clicking it opens a popup with a short explanation of what the section does, followed by an **Open in Manual:** link that switches to the Help tab and scrolls to the corresponding section.

Tooltips

Every label, input field, button, and checkbox in IDE4EEG has a tooltip describing what it does and (where applicable) the TOML key it controls. Hover over a widget for a second to see it.

In-app manual viewer

The right pane of the Help tab renders this `USER_MANUAL.md` file as HTML with a small CSS stylesheet that adapts to system light / dark mode. There are two search fields:

- **Above TOC**—narrows the left-pane TOC list by substring match against headings.
- **Above the manual text**—yellow-highlights every occurrence of the search term throughout the document, with up/down arrows to jump between matches and a counter showing the match count. The viewport scrolls to keep the current match in view (about a third from the top).

The text-search box widens the main window to ≥ 900 px when activated so long lines aren't broken across multiple visible lines (which would defeat scroll-to-match).

---

8. Helper applications

IDE4EEG works with four helper applications. The first three (Svarog, ConnectiVIS, empi) are **external**, launched on demand from the GUI as subprocesses; the fourth (MP Book Viewer) is **built-in** (an IDE4EEG Qt window), not external.

On first launch of a fresh install, SVAROG (with bundled empi), ConnectiVIS, and a Temurin JDK (latest LTS on macOS; 17 on Windows) are provisioned—they are treated as crucial parts of the package. The GUI shows a modal progress dialog with a Cancel button ~200 ms after the main window appears; CLI / batch mode (`ide4eeg --run config.toml`) runs the same flow silently with throttled stdout progress and auto-consents to the Java install.

When are the version-tracked helpers refreshed? On a fresh install or after an IDE4EEG upgrade (the `~/.obci/ide4eeg/helpers_fetched_version` sentinel is absent or no longer matches the running version), Svarog + ConnectiVIS are re-fetched to the latest CI build and **overwrite an older jar on disk**, including a hand-placed one at `~/.obci/svarog/`. Only a **repeat launch of the same release** (sentinel matches) is missing-only and leaves an existing jar untouched. Java is always install-if-missing, never force-refreshed. The one pin that survives a forced refresh is the `SVAROG_JAR` **environment variable** (checked first, and the refresh only ever writes into `~/.obci/svarog/`)—set it to hold a specific jar across upgrades. Per-tool errors are accumulated rather than aborting the chain—if ConnectiVIS fails to download, SVAROG and Java still install. See External tool paths for the path layout and per-row **Download recent** overwrite contract.

Svarog

Svarog is the open-source EEG signal viewer developed at the BrainTech Ltd. and the University of Warsaw. IDE4EEG launches Svarog whenever a synchronised, time-aligned interactive view is needed:

- **Step-result preview** (👁 buttons on the preprocessing panels)—split-view of before/after signal.
- **Bad channels review** (the Detect bad channels panel's ✎ decision button)—Svarog's `--select-mode bad_channels`.
- **ICA components review** (the ICA panel's ✎ decision button)—Svarog's `--select-mode ica_components`.
- **Mark detected artifacts review** (the Mark detected artifacts panel's ✎ decision button)—Svarog's `--select-mode bad_epochs`.
- **MP Book overlay**—when an empi `.db` book is alongside a `.fif` signal, Svarog can show the atom map atop the signal scroll.
- **Output-tab signal preview**—double-click a signal file in the tree.

Runtime requirements (Svarog 4.20):

- **Java:** JDK 11 or newer, any vendor (Temurin, Zulu, OpenJDK, Oracle); tested through JDK 25. Headless IDE4EEG runs that don't open Svarog need no Java at all. The first-launch flow auto-installs Adoptium Temurin—the **latest LTS cask** on macOS (`brew install --cask temurin`) and **JDK 17** on Windows (`winget install EclipseAdoptium.Temurin.17.JDK`)—in the GUI after one-time consent; CLI / batch mode auto-consents. On Linux (and macOS without Homebrew) the post-install dialog surfaces a manual-install hint with the adoptium.net URL.
- **Optional AppCDS speedup.** On JDK 13+, the Svarog launcher script writes an Application Class-Data Sharing archive next to the jar—`svarog-standalone-<version>-<jvm-tag>.jsa`—and reuses it on subsequent runs, saving ~380 ms (≈12%) off cold-start. JDK 11/12 skip AppCDS silently (JEP 350 dynamic CDS archives didn't ship until 13) and launch normally. The archive auto-regenerates when the jar is newer; the `<jvm-tag>` hashes the path to the `java` binary, so a JDK upgrade on a different install path generates a fresh archive automatically. AppCDS only kicks in via Svarog's launcher (or via IDE4EEG's Python wrapper, which opts into the same `.jsa` filename so it shares the cache for the same `(jar, JVM)` pair)—calling `java -jar svarog-standalone-*.jar` directly bypasses it and pays the full cold start every time.
- **Svarog ≥ 4.20.6 is required.** The bad-channel / bad-epoch / ICA-component review round-trip uses Svarog's text-format interface (the `-selection.txt` selection and `-tag.txt` artifact-mark files), which older jars—speaking only the legacy obci `.tag` format—cannot read. Because IDE4EEG drives Svarog by CLI

flag and Svarog hard-fails on an unknown flag, an older jar is not a degraded Svarog but a wrong one: auto-detection **skips it entirely and reports Svarog as not installed**, so every Svarog feature (signal preview, the MP-book overlay via `--book`, split-signal, review) is unavailable until you click **Download recent**. A jar pinned through `SVAROG_JAR` bypasses that filename check, but is still probed before a review—if its `java -jar … --help` output lacks `--select-mode`, the review **falls back to the MNE viewer** (a warning in the Run tab). (IDE4EEG's own MP **Book Viewer** window is built-in and Svarog-independent—see §8.4.)

The Config tab's **Download recent** button installs the latest CI build into `~/.obci/svarog/`. The bundled empi binary inside the SVAROG artifact is auto-extracted alongside the jar in the same step—there is no separate download for empi.

ConnectiVIS

ConnectiVIS is a 3D viewer for EEG source-reconstruction results: it shows dipole sources as oriented cones on the cortex and directed connectivity as coloured arrows between electrodes. IDE4EEG launches it automatically when dipole or connectivity results are produced, or via the **3D view** / **3D view all** buttons on the Output tab. It is auto-downloaded on first launch alongside SVAROG (see External tool paths); the Config tab's **Download recent** button refreshes it independently.

Runtime requirements (ConnectiVIS 2.0+):

- **Java:** JDK 11 or newer (any vendor). ConnectiVIS ships as a self-contained fat JAR built via `maven-assembly-plugin`; no separate dependencies need to be installed.
- **Building from source** (not normally needed—the Config tab's **Download recent** button installs a prebuilt JAR): Maven 3.6+. The Java package is `pl.edu.fuw.connectivis`, organised into five Maven modules (`cvis-model`, `cvis-io`, `cvis-animation`, `cvis-renderer`, `cvis-gui`); 3D rendering uses jogamp Java3D 1.7.2.

ConnectiVIS is the modernised rewrite of Trans3D / Glowa3D (CC Otwarte Systemy Komputerowe) for the Biomedical Physics Division, Faculty of Physics, University of Warsaw. The CLI accepts `.dat` connectivity matrices with bundled 10-20 electrode positions, or a custom `--electrodes montage.dat`.

Dipoles panel (right-side controls):

- **Size mode**: Uniform / Moment (nAm) / GoF (%) / Frequency (Hz)—scales each cone by the chosen attribute, normalised so the largest dipole renders at "slider × 1".
- **Colour mode**: same four options—maps the attribute through the dipole colormap, auto-stretched to `[min, max]`.
- **Size slider**: global multiplier (0–20).
- **Transparency slider**: cone fill (0 = solid, 100 = outline only). The outline is always opaque so direction stays visible.
- `data: min - max unit`: grey readout of the attribute range across loaded dipoles.

Signal arrows panel (connectivity):

- **Visibility threshold**: arrows below the slider are hidden. Auto-seeded at load to show roughly the 12 strongest arrows.
- **Arrow size / transparency**: global controls.
- Colour map stretches over min/max values; editable from the Colors window.

Scalp / Cortex panels: show/hide checkbox + transparency slider each. The cortex carries per-vertex Desikan-Killiany atlas colours from FreeSurfer.

Show panel: four checkboxes gating visibility of electrodes, dipoles, electrode name labels, and signal arrows.

Brain atlas legend (View menu): clickable list of cortex regions—clicking a region flashes it on the mesh for 3 seconds. "Plain cortex" toggle removes atlas colours.

Colors window (View menu): colormaps for signal arrows and dipoles, electrode colour, background colour.

Mouse and keyboard.

- Left-drag on 3D view: rotate. Mouse wheel: zoom.
- Arrow keys: rotate (← ↑ fine ~3°, → ↓ coarse 45°).

- `End`: front view. `Home`: side view. `+`/`-`: zoom.

File formats.

- `___dipoles.csv` / `___erp_dipoles.csv` / `___mxne_dipoles.csv` / `___mne_inverse_<method>_dipoles.csv` / `___lcmv_dipoles.csv`—dipole parameters with MNI-mm positions. ConnectiVIS reads columns by name; required: `ori_x/y/z` + `mni_x/y/z_mm` (preferred) or `pos_x/y/z` (fallback). Optional: `amplitude_nAm`, `gof_pct`, `frequency_Hz` for size/colour modes. The `mne_inverse` and `lcmv` files are written by default (`n_peaks=5`); set `n_peaks=0` to skip.
- `<base>___<tag>_<data_label>_<method>_<band>.dat`—inter-electrode connectivity matrices (square k × k, `;electrodes =` + `;band =` + `;title =` headers). One file per (method × band)—see §4.3 for the band table.
- `___electrodes.dat`—electrode positions in MNI mm (`name x y z` per line).
- `___brain.ply` / `___head.ply`—cortex and scalp surfaces (PLY with per-vertex RGB).

empi

empi is the GPU-accelerated Matching Pursuit decomposition engine (Różański 2024). IDE4EEG invokes it via subprocess for §3.8 MP decomposition; the binary ships inside the SVAROG artifact (Svarog's `mp/` subdirectory) and is auto-extracted in the same first-launch step that fetches Svarog—there is no separate empi download.

empi's CLI is fully documented in its README; IDE4EEG exposes its parameters under `[matching_pursuit]` in TOML and the MP Decomposition panel in the GUI. The Config tab's **Download recent** (Svarog row) refreshes the matching empi build alongside Svarog.

MP Book Viewer

The MP Book Viewer is a standalone interactive window for browsing Matching Pursuit decomposition results produced by empi. Unlike the other helpers, it ships built-in (`ide4eeg.analysis.mp_bookviewer_qt`) and runs as a Qt window inside the IDE4EEG process.

It displays Wigner time-frequency energy maps, original signal waveforms, and signal reconstructions from selected atoms.

Opening a file

Three ways to open an empi `.db` file:

1. **From the Output tab**—select an empi `.db` file in the file tree and click **Open in Book Viewer**.
2. **From within the viewer**—click **Open...** and choose a `.db` file.
3. **From the command line:**

```
python3 -m ide4eeg.analysis.mp_bookviewer_qt              # opens an empty viewer (use its Open…
 ↪  button)
python3 -m ide4eeg.analysis.mp_bookviewer_qt path/to.db   # opens directly
```

8.4.2 Layout

Four panels stacked vertically:

Panel	Content
Wigner map	Time-frequency energy density computed from Gabor atoms. Each atom contributes a 2D Gaussian blob. Atoms are shown as white dots; selected atoms have pink rings. Filtered-out atoms appear as small grey dots.
Signal	Original signal waveform (read from the `samples` table in the `.db` file).
Recon	Full reconstruction from the currently filtered atoms.
Selected	Reconstruction from only the manually selected atoms.

A colour bar next to the Wigner map shows the energy scale. The bottom status bar shows atom parameters when an atom is clicked.

8.4.3 Navigation

Action	Control
Next / previous **segment**	Toolbar ◀ / ▶ buttons, or **Left** / **Right** arrow keys
Next / previous **channel**	Toolbar ▲ / ▼ buttons, or **Up** / **Down** arrow keys
Zoom in / out	Scroll wheel (centred on cursor). Wigner map: zooms time and frequency; signal panels: time only.
Reset zoom	Double-click on the Wigner map, or press **Escape**
Change colour palette	Palette dropdown (jet, hot, viridis, inferno, gray)
Change energy scale	Scale dropdown: linear (raw values), log ($\log_{10}(1 + \text{energy})$), sqrt ($\surd$energy)

8.4.4 Atom selection

Click an atom dot on the Wigner map to toggle its selection:

- **Selected atoms** are marked with a pink ring on the map.
- The **Selected** panel at the bottom shows the waveform reconstructed from only the selected atoms.
- The status bar displays the clicked atom's parameters: iteration, frequency, centre time, scale, amplitude, energy, phase.
- Click **Clear sel.** to deselect all atoms.

Useful for selective signal reconstruction—pick specific time-frequency components to see how they contribute to the signal.

8.4.5 Atom filter bar

Click **Filter...** on the toolbar to show the filter bar. Enter criteria to restrict which atoms are displayed and used for the Wigner map and reconstruction:

Criterion	Description
Freq (min – max)	Keep atoms whose centre frequency is in this range (Hz).
Time (min – max)	Keep atoms whose centre time is in this range (s).
Energy ≥	Keep atoms with energy at or above this threshold.
Iter ≤	Keep atoms from the first N iterations only (higher-energy atoms are found first).

Click **Apply** (or press Enter in any field) to recompute the Wigner map. **Reset** clears all criteria. Filtered-out atoms are shown as faint grey dots on the map.

This is **nonlinear filtering**: the energy map is recomputed from the selected atom subset, not just masked. The reconstruction panel reflects only the filtered atoms.

8.4.6 Keyboard shortcuts

Key	Action
← / →	Previous / next segment
↑ / ↓	Previous / next channel
Escape	Reset zoom to full range
Enter (in filter field)	Apply filter

Appendix A. Installation

A.1 Python environment

IDE4EEG is developed on Python **3.14** (current target). The supported range is **Python 3.11–3.14**: 3.11 is the de facto floor (the input loader uses the stdlib `tomllib` module, added in 3.11) and 3.14 is what active development happens on. Tested on 3.12 and 3.14; 3.11 and 3.13 are expected to work but not currently exercised. Anything older lacks `tomllib` and will fail at import.

```
python3 -m venv .venv && source .venv/bin/activate
pip install ide4eeg                # lite - EEG core only, always succeeds
# or
pip install ide4eeg[video]         # adds OpenCV / PyAV / imageio / InsightFace / onnxruntime
# or
pip install ide4eeg[iclabel]       # adds onnxruntime for the ICLabel ICA selector (no video stack)
```

Use `[iclabel]` if you want ICLabel-based ICA component classification but not the video/facetag pipeline.

The lite install is the recommended starting point: it succeeds on every supported (Python, OS, arch) cell and fits in ~500 MB. Video processing (face detection + gaze artifact tagging) is opt-in. You can also install the video stack from inside the GUI later—Config tab → Tool Paths → Video stack → **Install all video tools** drives the same pip-install in a streaming-log dialog. L2CS-Net (the default eye-gaze backend) is always installed in-app via the **Download recent** button next to L2CS-Net, regardless of which scope you pick on the command line—PyPI bans direct VCS dependencies and L2CS only exists at a GitHub URL.

To launch:

```
ide4eeg                                # GUI, blank state (default)
ide4eeg myconfig.toml                  # GUI with config preloaded
ide4eeg --run myconfig.toml            # batch run, no GUI
ide4eeg --run myconfig.toml -t         # batch with full Python tracebacks
python -m ide4eeg                      # equivalent fallback if the
                                       # console script isn't on PATH
```

In GUI mode the optional positional path is just a shortcut for clicking *Load pipeline settings .toml* right after launch—it preloads the form so the user can review / edit before running. Batch mode (`--run`) requires a config path explicitly; there is no current-directory default.

Known wheel gaps

Some packages in the `[video]` extra have wheel gaps on specific (Python, OS, arch) cells. The lite install (`pip install ide4eeg`) is unaffected.

Cell	Affected	Resolution
Intel Mac + Python 3.14	`onnxruntime` (no cp314 x86_64 wheel)	Stay on lite install—facetag will be unavailable. To enable facetag here, downgrade to Python 3.13 (cp313 wheels exist). The in-app Install button reaches the same wheel-resolution failure but surfaces it in a friendlier dialog.
Intel Mac + PyTorch (ICLabel / L2CS gaze)	`torch` **has** a wheel (2.2.2, the last Intel-macOS build) but it is compiled for NumPy 1 and breaks against the required NumPy 2: `RuntimeError: Numpy is not available`. Because `mne_icalabel` prefers the PyTorch backend when it is present, installing torch here breaks **ICLabel** as well as L2CS gaze.	Use the **lite** build—ICLabel runs via ONNX Runtime and gaze via the InsightFace head-pose backend, neither of which needs PyTorch. For this reason the full Intel `.dmg` is not offered, and the in-app "Install L2CS gaze" button should not be used on an Intel Mac—it now warns and asks for confirmation there before starting, since pip itself reports success. Apple Silicon is unaffected (torch ≥ 2.3 supports NumPy 2).

Cell	Affected	Resolution
Apple Silicon, Windows, Linux (any arch, any supported Python)	none	Both lite and `[video]` install scopes work directly. Linux + Python 3.14 was listed here until 2026-07-31 (`insightface` / `stringzilla` built from sdist and needed a compiler); upstream now ships a pure-Python `insightface` wheel plus cp314 wheels for `stringzilla` / `simsimd`, so no build tools are required.

Development install (from a checkout)

```
git clone https://gitlab.com/fuw_software/ide4eeg.git
cd ide4eeg
python3 -m venv .venv && source .venv/bin/activate
pip install -r requirements.txt    # = pip install -e .[video]
ide4eeg                             # editable -- ide4eeg/*.py edits go live
```

A.1.1 Desktop bundles (Windows installer / macOS .dmg)

The releases page also offers ready-made bundles that need no Python at all: a Windows installer and portable zip, and macOS `.dmg` images. They ship their own interpreter and dependencies, so nothing above applies to them.

macOS blocks the first launch. Getting past it takes four clicks and no Terminal:

1. Drag the app from the `.dmg` window to **Applications**, then eject the `.dmg`.
2. Double-click the app. macOS refuses to open it—click **Done**. This step is not optional: the button in step 3 appears only after a blocked attempt.
3. Open **System Settings → Privacy & Security**, scroll down to the message naming the app, click **Open Anyway**, and confirm with your password or Touch ID.
4. Double-click the app again. It opens, and macOS does not ask again.

The refusal in step 2 announces itself differently depending on the macOS version and how far the launch got. These all mean the same thing, and none of them indicates a defect in the app:

- *"…cannot be opened because Apple cannot check it for malicious software…"*
- *"Launch error—See the py2app website for debugging launch issues"*—this one **is not an application crash**, despite reading like one. The bundled app framework reports every startup failure with one generic dialog, so a security block and a real defect look identical here.
- *"…is damaged and can't be opened. Move it to the Trash"*—the app is **not** damaged.

The reason: the `.app` is ad-hoc signed but **not notarized** (notarization requires paid Apple Developer Program enrollment, which the project does not have). Downloading the `.dmg` in a browser attaches macOS's `com.apple.quarantine` flag to it, and mounting propagates it to every one of the ~21 000 files inside. While that flag is set, macOS runs the app through **App Translocation**—a randomised read-only copy—where Gatekeeper stops it before Python starts. The flag decides this, not the app's location: the block behaves identically from the mounted `.dmg`, from `/Applications` and from the Desktop, so moving the app around does not help.

If you would rather clear the flag directly than click through, one command on the disk image—*before* mounting it—covers the whole bundle:

```
xattr -d com.apple.quarantine ~/Downloads/IDE4EEG-lite-0.9-arm64.dmg
```

If the image is already mounted, eject it and mount it again; the flag reaches the contents at mount time. On an app that has already been copied out, clear it on the copy instead:

```
xattr -dr com.apple.quarantine /Applications/IDE4EEG.app         # full build
xattr -dr com.apple.quarantine /Applications/IDE4EEG-nogaze.app  # lite build
```

On **Windows**, SmartScreen shows *"Windows protected your PC"* on first run of the installer; click **More info → Run anyway**. Code signing is pending on a future release.

A.2 Java + helper apps

IDE4EEG uses three external helper applications (Svarog, ConnectiVIS, empi) that are JVM- or native-binary-based.

- **Java 11 or newer** (any vendor—Temurin, Zulu, OpenJDK, Oracle; tested through JDK 25) is required for **Svarog** and **ConnectiVIS**. JDK 13+ additionally enables AppCDS in Svarog for ~12% faster cold-start—see Svarog for the full Svarog runtime spec and ConnectiVIS for ConnectiVIS's. Headless CLI runs (`ide4eeg --run myconfig.toml`) that don't open Svarog or ConnectiVIS are the one path that doesn't need Java at all.
- The **Svarog / ConnectiVIS / empi** binaries themselves are fetched on demand by the GUI's **Download recent** button (Config tab); no manual install needed beyond the JVM. See Chapter 8 for what each one does and External tool paths for the install / TLS-troubleshooting details.

A.3 Video processing libraries (PyTorch + L2CS)

The default `requirements.txt` (`[video]`) install ships face detection + the InsightFace head-pose backend, which serves as the *fallback* gaze estimator. **L2CS-Net** (true per-eye gaze direction) is the **configured default backend** but ships separately—install it with the **Download L2CS for gaze detection** button on the Config tab. When L2CS isn't installed, gaze detection automatically falls back to InsightFace head-pose.

The button installs PyTorch + torchvision + the `l2cs` Python package + the Gaze360 weights checkpoint (~96 MB, MIT-mirrored from Ahmednull/L2CS-Net) in one shot. Total install ~500 MB—the CPU PyTorch wheel, on every platform (the installer pins the CPU index on Linux so a bare `pip install torch` can't drag in the ~2 GB CUDA build IDE4EEG never uses). There is no in-app uninstaller on the L2CS row; to reclaim the space either click **Uninstall video tools** in the Video-stack section header (see §1.2), or run `pip uninstall torch torchvision l2cs` and delete the Gaze360 weights file (`~/.obci/ide4eeg/models/L2CSNet_gaze360.pkl`).

See §3.10 Gaze for the choice between the two backends.

Appendix B. Supported EEG formats

Format	Input file	Companion files required
BrainTech	`.raw`	`.xml` required; `.tag` optional (events—without it, resting-state analysis only)
BrainVision	`.vhdr`	`.vmrk`, `.eeg`
MNE-FIF	`.fif`	*(none)*
EEGLAB	`.set`	`.fdt` if the data is stored separately (single-file `.set` needs none)
EDF / EDF+	`.edf`	*(none)*—annotations read automatically
BDF / BDF+	`.bdf`	*(none)*—annotations read automatically

Channel-type assignments differ per format—see Channel type inference for the layered inference pipeline. For polysomnographic mixed-modality recordings (PSG, MASS, Sleep-EDF, Physionet-PSG), EDF/BDF channels default to `eeg` in MNE; IDE4EEG's overlay re-runs `chtype_heuristic` on those defaults so EOG/EMG/ECG/respiration/oximetry channels are correctly typed.

To add a new format, see Appendix D.2.

Appendix C. Complete config.toml reference

Below is a fully-commented `config.toml` showing all parameters—both the commonly-set ones and the hidden defaults. Copy as a starting point and remove or adjust as needed.

> **Note:** a few values in the block are illustrative **examples**, not defaults—notably `input_path` (a sample recording path) and `segmentation.mode = "epochs"` (the shipped default is `"rest"`). Adjust them to your own data.

```toml
# <<< GENERATED: scripts/gen_appendix_c.py (do not edit by hand) >>>
# ======================================================================
# IDE4EEG -- Complete Configuration Reference
# ======================================================================

# --- Paths and general settings ---
input_path = "examples/dipole_spindles/dipoles_example.obci.raw"
output_path = "None"                    # "None" = save next to input file
segmentation = { mode = "epochs", reject_by_annotation = true }   # mode: "rest"/"epochs";
 ↪  reject_by_annotation: rest-mode PSD omits marked spans (§4.5)
overwrite_output = true

# --- Interactive reviews (GUI only; no effect on --run / batch) ---
# allow_manual_modifications = false    # master switch: may a review CHANGE the result? (§2 Other
 ↪  Options)
# manual_review_timeout_min = 480       # how long a review window may stay open; 0 = no limit

# --- Preprocessing toggles ---
# EEG artifacts runs by default; disable it by removing "eeg_artifacts"
# from preprocessing.step_order (no top-level toggle).
prepare_ica = false
prepare_video_artifacts = false
prepare_mp_filter = false
# facetag_frame_skip = 3              # process every 3rd frame (faster)
# facetag_downscale = 1.0             # 1.0 = full resolution; 0.5 = half (faster)
# facetag_backend = "insightface"     # only "insightface" is supported
# facetag_gaze_method = "l2cs"        # "l2cs" (default) or "insightface"
# facetag_detection_skip = 3          # re-detect faces every Nth processed frame
# trim_start = 0.0                    # crop signal start (seconds)
# trim_end = ""                       # crop signal end ("" = full signal)

# --- Analysis toggles ---
# MP decomposition is enabled by adding "mp_decomposition" to
# preprocessing.step_order - there is no top-level toggle.
prepare_eeg_profiles = false
prepare_connectivity_analysis = false
prepare_dipole_fitting = false
# mne_catalog = ["erp_butterfly", "cluster_permutation"]  # external analyses

# --- Signal setup ---
electrodes_layout = "native (from file)" # default sentinel; replace with e.g. "standard_1005"
re_reference = []                       # default = no re-referencing; e.g. ["M1","M2"] / "average" /
 ↪  "None"

# --- Advanced: signal trimming (hidden defaults) ---
# resample_freq = 0                    # Hz; 0 keeps original; set to e.g. 256 to downsample
# cover_time = None                    # deprecated, ignored
# threshold_time = None                # deprecated, ignored

# --- Parallelism ---
[parallelism]
n_jobs = 0                              # default = auto (physical cores); 1 = sequential


[choosing_channels]
dropped_channels = []                   # default = none; e.g. ["Audio","Sample_Counter","Photo"] to
 ↪  drop hardware-aux channels
selected_channels = "all"

# --- Advanced: channel quality (hidden defaults) ---
# The live bad-channel detector is the PREP-aligned [choosing_channels.prep]
# sub-block; its hidden defaults are documented in Appendix E.2.
```

```toml
[filters]
plot_filt = false
method = "iir"                          # "iir" or "fir"
highpass_freq = 0.5                     # Hz; default 0.5, 0 = off. Common: 0.1, 0.5, 1.0
lowpass_freq = 0                        # Hz; 0 = off (default). Common: 30, 40, 100
notch_freq = 50                         # Hz; default 50 (EU/Asia), 60 (Americas); 0 = off
notch_harmonics = true                  # also remove integer harmonics up to Nyquist (default ON)
# save = false                          # write the <base>-filtered-raw.fif snapshot


[eeg_artifacts]
# [EEG artifacts] panel -> subpanel [Amplitudes: P2P and slope] (tag-first).
notch_hz          = 50.0     # mains notch (0 / None = off); + notch_harmonics
notch_harmonics   = true
floor_rel_k       = 1.0      # adaptive floor = k x channel median p2p; 0 = use absolute uV floors
min_amplitude_uv  = 50.0     # absolute p2p floor (uV); used only when floor_rel_k = 0
crossover_ms      = 50.0     # config-only: slope band below / p2p above
max_scale_ms      = 300.0    # config-only: coarsest p2p detection window
amp_floor_window_ms = 0.0    # config-only: amplitude-floor window; 0 = inherit max_scale_ms
min_windows       = 20       # config-only: n_eff gate, both detectors
p2p_enable        = true
p2p_z_threshold   = 5.0      # robust-z cutoff for p2p
p2p_ceiling_uv    = 0.0      # abs gross-excursion ceiling (uV); 0 = off
slope_enable      = true
slope_z_threshold = 5.0      # robust-z cutoff for slope
slope_min_amplitude_uv = 50.0  # absolute slope step anchor (uV); used only when floor_rel_k = 0
slope_min_samples = 2        # config-only: first-difference floor
hf_enable         = true     # muscle / EMG high-frequency band power
hf_lo_hz          = 90.0     # EMG band lower edge (uV; above neural band)
hf_hi_cap_hz      = 250.0    # upper edge; effective = min(this, 0.45*fs)
hf_z_threshold    = 5.0      # robust-z cutoff for hf (one-sided)
hf_window_s       = 0.5      # config-only: HF power window (non-overlapping)
flat_enable       = true     # relative flat-span detector (low tail of p2p)
flat_fraction     = 0.2      # flat when ptp < this × median(ptp) per channel
flat_window_s     = 0.2      # config-only: ptp window for flatness
flat_min_duration_s = 0.5    # config-only: min flat run to mark (s)


[ICA_EOG]
method             = "picard"          # "picard", "infomax", or "fastica"
# selector: the "iclabel" shown here is the out-of-box default ONLY WHEN an
# ICLabel backend (onnxruntime or pytorch) is installed; a backend-free install
# defaults to "find_bads" instead (heuristic EOG/ECG/muscle, needs no backend).
# The value is backend-conditional at import time -- see §3.9.
selector           = "iclabel"         # "iclabel", "find_bads", "both", "none"
fit_highpass_hz    = 1.0               # Winkler 2015 fit-time HP (fit copy only); 0 = off
fit_lowpass_hz     = 0.0               # fit-time low-pass (fit copy only); 0 = off
fit_notch_hz       = 0.0               # fit-time notch (fit copy only); 0 = off
fit_notch_harmonics = true             # also remove harmonics (default ON)
reject_uv          = 500               # p2p µV - drop crazy segments pre-fit
reject_tstep       = 2.0               # window (s) MNE scores p2p against
n_components       = "rank"            # "rank" | int | float(0..1)
# iclabel_keep: the classes to KEEP -- every other class is removed.  NB the
# GUI's ICLabel box is a REMOVE-list, so it shows the COMPLEMENT of this: the
# default below appears there as muscle/eye/heart/line_noise/channel_noise
# ticked.  Hand-editing this key means writing what to keep -- see §3.9.
iclabel_keep       = ["brain", "other"]
iclabel_min_prob   = 0.0
# --- Only used when selector ∈ {find_bads, both} ---
find_bads_eog      = true
```

```
find_bads_eog_ch    = []
find_bads_ecg       = true
find_bads_muscle    = true
# --- Advanced ---
decim               = 1                 # fit subsampling factor
random_state        = 42
save                = false             # write audit tree + cleaned -raw.fif


[rest]
rest_duration = [0, ""]                 # [start_s, end_s]; "" = full signal
window_length = 20                      # seconds
whole_signal = false                    # true = one window spanning the entire signal


[epochs]
start_offset = -0.3                     # seconds before event
stop_offset = 0.7                       # seconds after event
epochs_baseline = "None"                # "None", [start, end], or ["None", "None"]
# --- Advanced (hidden defaults) ---
# (The epoch cut uses reject=None; bad segments are derived afterwards by
#  [Mark detected artifacts] aggregating the upstream artifact marks, §3.12.)

    [epochs.tags]
    selected = ["event_name_1", "event_name_2"]

[segment_rejection]
# [Mark detected artifacts] owns no DETECTOR (see §3.12), but does two
# things: it CONDITIONS the marks (floor/consolidate/dilate -> the final
# marks every consumer reads, always) and optionally DROPS via 'Drop marked'
# + its policy - the contaminated-fraction gate + the G1 run-length trigger
# (auto_reject in event mode, drop_in_continuous in continuous).
auto_reject = true                      # master gate; false = mark-only
drop_in_continuous = false              # continuous mode: also drop contaminated windows
artifact_threshold = 0.3                # per-channel contaminated-% gate (GUI %)
artifact_min_run_ms = 200.0             # G1: drop a segment if any channel's
                                        #   longest contiguous contaminated run
                                        #   reaches this; 0 = off (fraction-only)
enable_fraction = true                  # GUI: derived from the % field > 0
enable_run = true                       # GUI: derived from the ms field > 0
                                        #   (consensus_frac = 0 => > 0 => any channel)
consensus_frac = 0.0                    # quorum frac (0-1; 0 = any; GUI shows %)
# Per-type mark conditioning (0 = that type's no-op; defaults below): min_mark_ms
# = duration floor widening short marks; dilate_ms = symmetric pad; consolidate_ms
# / consolidate_enable = merge marks separated by less than the gap. Shared by the
# epoch drop and the rest-mode continuous PSD (§4.5). Keys: p2p/slope/hf/flat/gaze.
min_mark_ms       = {p2p = 100.0, slope = 100.0, hf = 100.0, flat = 100.0, gaze = 100.0}
dilate_ms         = {p2p = 20.0, slope = 20.0, hf = 20.0, flat = 50.0, gaze = 50.0}
consolidate_ms    = {p2p = 100.0, slope = 100.0, hf = 100.0, flat = 100.0, gaze = 100.0}
consolidate_enable = {p2p = true, slope = true, hf = true, flat = true, gaze = true}


[connectivity]
methods = ["dtf", "coh"]            # list (or comma-separated string); default
mvar_method = "yw"                  # yw, ns, vm
mvar_order = 0                      # 0 = auto (Akaike)
resolution = 100
channels = "all"                    # default = all EEG channels; subset list to restrict, e.g.
 ↪  ["Fz","Cz"]
short_time = false                  # true = time-resolved sliding-window connectivity
st_window = 0                       # sliding-window length [s]; 0 = signal length / 5
st_overlap = 0                      # sliding-window overlap [s]; 0 = signal length / 10
```

```
significance = false                # true = surrogate / bootstrap significance test
sig_reps = 100                      # surrogate / bootstrap replications
sig_alpha = 0.05                    # significance level (α)


# --- EEG profiles ---
# [eeg_profiles]
# mode = "p2p"                      # "count", "percentage", "p2p", "power"
# channel = 0
# (Bin width is always the MP segment length - not configurable.)

# --- Dipole fitting ---
# [dipole_fitting]
# ref_channel = "average"
# montage = "standard_1005"
# max_iterations = 0                # 0 = all atoms
# min_gof = 0                       # 0 = no constraint
# trans_path = ""                   # MRI<->head transform; "" = auto-discover

# --- MP decomposition (enabled via preprocessing.step_order) ---
# [matching_pursuit]
# (decomposition parameters: algorithm, iterations, ... - see §3.8)
#     [matching_pursuit.outputs]
#     atom_stats_csv = false          # write per-atom statistics CSV
#     atom_histograms = false         # write atom-parameter histogram plots

# --- MP filter (preprocessing) ---
# Internal decomp + reconstruction; independent of [matching_pursuit].
# [mp_filter]
# window_length = 0                 # 0 = inherit from [rest].window_length
# [mp_filter.decomp]
# algorithm = "smp"
# iterations = 30
# explained_energy = 0.99
# [mp_filter.filter]
# mode = "keep"                     # "keep" or "remove"
# freq_min = 0.5                    # "osc. EEG" preset defaults
# freq_max = 49
# scale_min = 0.1

# --- Per-step signal snapshots (the save drawer) ---
# Each of these steps can write its intermediate signal to
# saved_steps/<base>-<suffix>-raw.fif.  Each is a free toggle, off by
# default.  The ICA_EOG and filters blocks above carry their own save
# key; the rest need a one-key section of their own.
# [trim]
# save = false
# [resample]
# save = false
# [bad_channels]
# save = false
# [reference]
# save = false
# [mp_filter]
# save = false

# --- Preprocessing step order ---
# [preprocessing]
# step_order = [...]                # omit = canonical default order
#
# Step-ordering advice is FIXED IN CODE and is not configurable.
# It warns, never blocks.  The shipped recommendations are:
```

```
#   bad_channels before ica
#   reference before filtering
# <<< END GENERATED >>>
```

Appendix D. FAQ

D.1 General

How do I check my Python version?

```
python3 --version
```

See Appendix A.1 for the supported range.

I get `ModuleNotFoundError`

Make sure the virtual environment is activated. You should see `(venv)` or `(.venv)` at the start of your terminal prompt.

D.2 File formats

IDE4EEG does not recognise my file

Check that:

1. The file extension is `.raw`, `.vhdr`, `.fif`, `.set`, `.edf`, or `.bdf`.
2. Required companion files are present (e.g. `.raw` needs `.xml`; `.tag` is optional—without it, resting-state analysis only).
3. The `input_path` in your config points to the correct file or directory.

How do I add a new format?

Add a reader function in `ide4eeg/input/input.py` that returns `(mne_signal, events_desc_id)` where `mne_signal` is an `mne.io.RawArray` with EEG channels and a STIM channel, and `events_desc_id` is a dict `{'event_name': integer_id, ...}`.

Then add the extension to the module-level `SIGNAL_EXTENSIONS` tuple (used by `find_file_paths`) and a dispatch case in `read_file`. Adding a call to `_refine_channel_types(raw)` at the end gives the new format the same channel-type heuristic used for BrainTech and EDF.

D.3 Preprocessing

How do I define custom filters?

Filters are specified by **cutoff frequency**, not by name. In the GUI use the Filters panel on the Preprocessing tab; in TOML set `highpass_freq`, `lowpass_freq`, and `notch_freq` (Hz; `0` disables each) and optionally `method = "iir"` (default) or `"fir"` inside `[filters]`. Transition bandwidths and filter order are derived automatically by `design_iir_filter` in `ide4eeg/preprocessing/channels_and_signal.py` (see §3.6 and Appendix D.5 for the design details). There is no per-filter Python dict to edit.

How do I select which events to include?

In the GUI, load a file and check the desired events in the **Events** panel (Preprocessing tab, Segmentation setup section). For TOML configs:

```toml
[epochs.tags]
selected = ["event_1", "event_2", "event_3"]
```

Where are the clean signals saved?

The cleaned epochs are always written to `preprocessing/saved_signals/` as `-epo.fif` files—this is the primary pipeline output. Intermediate signals from individual preprocessing steps are written to `preprocessing/saved_steps/` only when their per-step **Save** checkbox is checked.

I checked a step's Save checkbox but I don't see any output. Why?

Save checkboxes default to off. When checked, the step's intermediate output (transformed signal, detection plots, ICA components, etc.) is written under `preprocessing/` after the next pipeline run. If you ran the pipeline before checking the box, run it again to produce the saved output.

D.4 Analysis

All epochs were rejected—what should I do?

Your rejection criteria may be too strict. Try:

- Increasing `artifact_threshold` and/or raising `artifact_min_run_ms`, so brief marks no longer drop a segment. Set `artifact_min_run_ms = 0` to disable the run-length rule entirely.
- Unticking **"Drop marked"** (`[segment_rejection] auto_reject = false`) for mark-only review—the artifact marks stay as tags/annotations but nothing is auto-dropped.
- Relaxing the upstream EEG-artifact detectors (raise the `*_z_threshold` cutoffs or the `*_amplitude_uv` anchors), since `[Mark detected artifacts]` only aggregates their marks.
- Reviewing the Run-log drop summary to see which channels and epochs were rejected (the per-segment statistics DataFrame is not written to disk).
- Reviewing filters and ICA settings.

How do I customise the hidden default parameters?

Add them directly to your `config.toml`. Top-level parameters (like `resample_freq`, `cover_time`) go at the root level. Section-specific parameters go inside their section (e.g. `auto_reject` inside `[segment_rejection]`). See Appendix E for the full list, or Appendix C for a complete config template.

Cluster test plots show no clusters, but the log mentions some

Detailed cluster information is saved in a `.txt` file in the `MNE/` folder.

D.5 Resampling & filtering

How does filtering work?

IDE4EEG supports two filtering methods:

- **IIR (Butterworth / Chebyshev II)**—default. Fast, sharp rolloff. Filter order and transition bandwidths are computed automatically from the cutoff frequency and the signal's sampling rate using `scipy.signal.iirdesign`. Stored in second-order sections (SOS) format and applied zero-phase via `sosfiltfilt`. The forward+backward pass squares the filter's magnitude response—a `-3 dB` design becomes `-6 dB` at cutoff—and doubles the effective filter order. Filter-response plots show the single-pass design, not the realised doubled response.
- **FIR (MNE)**—alternative. Hamming-windowed sinc design with automatic filter length. Always stable, linear phase, zero group delay. Slower than IIR but no stability concerns at any sampling rate.

If a cutoff exceeds the Nyquist frequency, that filter is skipped with a warning rather than producing an invalid design.

D.6 Video artifacts

For a complete description of the pipeline, backends, gaze methods, and all parameters, see §3.10 Gaze.

What are exclude faces?

When recording EEG from children, a parent (or another adult) is often visible in the video. When the child turns away from the screen, only the parent's face may be detected and falsely matched as the subject, producing incorrect "looking" reports.

The **Exclude faces** directory lets you specify photos of people to reject. Any detected face that is closer to an exclude reference than to the subject reference is automatically rejected. Leave the field empty to disable exclusion.

How can I speed up video artifact detection?

Two acceleration options:

- **Frame skip** (`facetag_frame_skip`): process every Nth frame instead of every frame. Skipped frames inherit the result of the last analysed frame. At 30 fps, `facetag_frame_skip = 5` gives ~5× speedup with minimal accuracy loss.
- **Downscale** (`facetag_downscale`): shrink each frame before face detection. `facetag_downscale = 0.5` (half resolution) roughly halves processing time per frame. Avoid values below 0.3 if the infant's face is small in the frame.

Both options can be combined. For example, `facetag_frame_skip = 3` with `facetag_downscale = 0.5` gives roughly 6× overall speedup.

How do I create reference photos?

Reference photos are images of the subject looking directly at their **task target**—the stimulus screen or fixation point they watch during the recording—*not* at the camera, which normally sits above or beside it. You need 3–5 such images. See What the reference photos actually define for why the direction matters.

1. **Select from video** (recommended): in the Preprocessing tab, click **Reference faces** on the Gaze panel. The tool opens the video, lets you scrub through frames and click on the subject's face to save it. Use `</>` to step one frame, `<</>>` to jump to the next frame with a detected face. Saved faces appear as cropped thumbnails below the video; click a thumbnail to remove it.
2. **Manual**: place 3–5 JPEG/PNG images in a folder. Each image must contain exactly one face (the subject) looking at their **task target**—the stimulus screen or fixation point they watch during the recording, not the camera (which usually sits above or beside it). The reference photos *define* what counts as looking-on-task: `check_gaze` scores every frame by its angle from these vectors, within `facetag_max_angle`.

By default, picked photos are saved into `<output>/IDE4EEG_OUT_<base>/preprocessing/reference_faces/` so they live alongside the rest of the per-recording results (and `exclude_faces/` for the exclude set). Use **Change folder** to override the location.

How do I review detections?

Click the 🔒/✏ **decision button** on the **Detect gaze artifacts** panel header (the step has no 👁 eye—it changes no samples). This opens the Review Video Artifacts window (auto-loading any previous results), showing the video and EEG side-by-side with the detected intervals and Confirm / Dismiss / Save Changes actions.

Which signal is shown? The review window shows the most-recent upstream snapshot—i.e. the signal after any preceding preprocessing steps that have their **Save** checkbox enabled (filtering, resampling, bad-channel rejection, montage, ICA, etc.). When you've already run the pipeline once with `[filters].save = true`, the gaze review canvas shows the *filtered* signal—same data the gaze detector saw. When no upstream snapshot exists yet, the review window falls back to the raw input file. Freshness is checked the same way as the eye button—a stale `[cfg:<hash>]` is regenerated.

EEG signal plot colours.

- **Pink/red shaded regions**—accepted not-looking intervals (confirmed artifacts).
- **Grey shaded regions**—dismissed detections (intervals marked as false positives).
- **White dashed vertical line**—current playback position.
- **Coloured traces**—individual EEG channels (colours distinguish channels, not gaze status).

Controls.

- **Run Detection**—run gaze analysis on the video using the reference photos. Detection respects trim settings and intervals appear in the list as they are found.
- **Load Previous Results**—load a previously saved JSON results file.
- **Confirm Artifact**—mark selected intervals as true artifacts.
- **Dismiss Detection**—mark selected intervals as false positives.
- **Go to Interval**—jump the video and EEG view to the start of the selected interval.
- **Save Changes**—save the current interval statuses to a JSON file and a Braintech/Svarog `.tag` file.

Double-click any interval row to jump to its start time.

D.7 GUI tips

- Hover over any label or input field to see a tooltip describing the parameter.
- The Run tab calls the same code as `ide4eeg`, so CLI and GUI results are identical.
- The **Output** tab is a results browser with a navigation row at the top—see Browsing the tree.
- The **View > Light theme / Dark theme / System theme** menu lets you switch palettes; **System theme** follows the OS appearance.
- The **Go** menu jumps to a specific tab by keyboard.

Appendix E. Advanced / hidden parameters

The following parameters have sensible defaults set internally in `check_and_prepare_config` (`input.py`). They are not included in the default `config.toml` but can be added to override the defaults. Parameters that belong to a specific section (`[choosing_channels]`, `[epochs]`, etc.) must be placed in that section; top-level parameters go at the root.

E.1 Signal trimming

Top-level parameters.

Parameter	Type	Default	Description
`resample_freq`	int	`0`	Target sampling frequency (Hz). 0 = keep original rate.
`cover_time`	float or `None`	`None`	**Deprecated**—ignored. Use `trim_start` / `trim_end`.
`threshold_time`	float or `None`	`None`	**Deprecated**—ignored. Use `trim_start` / `trim_end`.

When both `trim_start` and `trim_end` are unset, the signal passes through uncropped.

E.2 Channel quality (advanced)

All five detectors and their shared pre-conditioning live under the `[choosing_channels.prep]` sub-block. Every key has a literature-backed default (PREP-published where the criterion comes from PREP; `clean_flatlines`-derived for the flatline-duration detector); only edit what you have a recording-specific reason to change.

Pre-conditioning (shared by all five detectors)

Parameter	Default	Description
`hp_freq_hz`	`1.0`	High-pass cutoff applied to the working copy before any statistic is computed. DC drift dominates raw amplitude and destroys correlation.
`notch_hz`	`50.0`	Line-noise notch frequency in Hz; `null` (or `0`) = no notch. 50 Hz (Europe / most of Asia / Africa / Australia); 60 Hz (Americas / parts of Asia / Japan). Inherited from `filters.notch_freq` at config-load when that is set. The GUI dropdown only offers `off / 50 / 60`; CLI users hand-editing the TOML to a non-50/60 frequency (e.g. 47 or 100 Hz) get a `WARNING` log line on the next GUI populate naming the snap back to 50.
`notch_harmonics`	`true`	When ON, the notch removes the fundamental AND every integer harmonic up to Nyquist (50/100/150/… or 60/120/180/…).
`max_iterations`	`1`	Refinement passes after the first detection (0–4). Each pass drops the channels found so far and re-runs the three cross-channel detectors on the cleaner reference, surfacing borderline channels the obvious outliers masked. Stops early on convergence. `0` = single pass; `1` = recommended default; `4` = PREP's cap. (Old `iterate_once = true/false` maps to `1`/`0`.)

Detector 1—Absolute amplitude checks

A bundled if-then conditional: *if* the median SD across channels is plausible, *then* flag each channel whose SD is below the dead-amp floor. See §3.3 prose for the design rationale. The GUI surfaces this as one panel with a master checkbox; the TOML keeps `enable_flat` and `enable_recording_amp_check` as two independent keys for back-compat (the GUI ties them together via a single bool—saving normalises them to the same value).

Parameter	Default	Description
`enable_recording_amp_check`	`true`	If-clause master. Both this AND `enable_flat` must be `true` for the bundled GUI checkbox to render ON.
`enable_flat`	`true`	Then-clause master. Bundled with `enable_recording_amp_check` in the GUI.
`recording_amp_check_min_uV`	`0.5`	Lower bound of plausible median channel SD (µV). Below this, real scalp EEG is implausibly small.
`recording_amp_check_max_uV`	`200.0`	Upper bound. Above this is implausibly large—likely a wrong-units anomaly. Set both bounds wide (e.g. `0` and `1e9`) to disable the gate and run the per-channel SD check unconditionally.
`flat_sd_threshold_uV`	`1e-3`	Per-channel SD floor in microvolts (= 10^{-9} V, PREP MATLAB's literal `findNoisyChannels.m:289` value). Anything with robust SD below this is treated as flat *provided the median-SD sanity gate passes*. See docs/literature-reviews/EEG_bad_channels.pdf for the threshold rationale.

Detector 2—Flatline duration (eps-relative)

Mirrors EEGLAB `clean_flatlines.m`. Calibration-insensitive by construction; complements Detector 1.

Parameter	Default	Description
`enable_flatline`	`true`	Enable the duration-based flat detector. Recommended ON.
`flatline_max_jitter`	`20`	How many multiples of machine precision count as "no change" between adjacent samples. `machine_precision = np.finfo(dtype).eps`; on float64 the effective threshold is ~4.4×10^{-15} V at the default 20—numerical-noise level, true only for bitwise-identical samples (silent ADC, railed channel, post-rereference collision). EEGLAB `clean_flatlines.m` default; almost never needs changing.
`flatline_max_duration_s`	`5.0`	Maximum contiguous flat-run duration tolerated, in seconds. EEGLAB `clean_flatlines.m` default. Shorten (e.g. 1 s) on brief event-locked recordings; lengthen (e.g. 10 s) on long resting-state recordings where a brief flat stretch is not yet pathological.

Detector 3—Amplitude outlier (PREP: "Robust deviation")

Parameter	Default	Description
`enable_deviation`	`true`	Enable the deviation detector. Recommended ON.
`deviation_z_threshold`	`5.0`	Robust `abs(z)` threshold across channels for the per-channel robust SD. Default. Lower = more sensitive (more false positives).

Detector 4—Windowed correlation (max over other channels)

Parameter	Default	Description
`enable_correlation`	`false`	Enable the windowed-correlation detector. Off by default—on standard 19-ch 10-20 montages the peripheral ring naturally carries r ≈ 0.7-0.9, so it over-flags; enable it (and tune `correlation_threshold`) for high-density / research montages.

Parameter	Default	Description
`correlation_threshold`	`0.3`	Minimum `abs(r)` with any other channel that counts as "OK" for a given window. Below PREP/pyprep's 0.4—fewer false positives on standard 19-ch 10-20 montages, whose peripheral ring (T7/T8/F7/F8/O1/O2/P7/P8) naturally carries r ≈ 0.7-0.9. Raise toward 0.7 on low-density / pediatric / high-impedance recordings; lower toward 0.2 if it over-flags.
`correlation_window_seconds`	`1.0`	Window length for the windowed correlation pass.
`bad_window_fraction_threshold`	`0.05`	Maximum fraction of windows that may fall below the correlation threshold before a channel is flagged. Default 5 % (raised from 0.01 so a few transiently decorrelated windows don't condemn a channel).

Detector 5—High-frequency noise ratio

Parameter	Default	Description
`enable_hf_noise`	`true`	Enable the HF-noise detector. Recommended ON.
`hf_noise_z_threshold`	`5.0`	Robust `z` threshold across channels of the HF/broadband SD ratio. Only positive deviations are flagged (high HF is bad; low HF is normal).
`hf_lower_hz`	`50.0`	Lower cutoff of the "high frequency" band for the HF-noise ratio. Lower this only if your recording is bandlimited (e.g. 30 Hz lowpass already applied).

Top-level (sibling block `[bad_channels]`):

Parameter	Default	Description
`bad_channels.save_plots`	`false`	**Currently dormant**—the PREP-aligned detector does not write diagnostic plots; the toggle is reserved for a planned follow-up. Use the Run-log per-channel diagnostic table (one line per channel per criterion with `[FLAGGED]` markers) in the meantime. GUI label: "Save detection plots".

Default TOML block

```toml
[choosing_channels.prep]
# Pre-conditioning applied to an internal copy before detection.
hp_freq_hz = 1.0
notch_hz = 50.0           # 50 Hz Europe / 60 Hz Americas; null = off
notch_harmonics = true
max_iterations = 1        # refinement passes (0-4); 0 = single pass

# Recording-level amplitude sanity check (median σ_M envelope).
# If the median is outside [min, max] µV, the SD-floor "flat" check
# is auto-disabled for the run (calibration anomaly detected).
enable_recording_amp_check = true
recording_amp_check_min_uV = 0.5
recording_amp_check_max_uV = 200.0

# Detector 1 - flat (SD floor, µV-absolute, calibration-gated).
enable_flat = true
flat_sd_threshold_uV = 1e-3   # = 10^-9 V; PREP MATLAB literal value
```

```
# Detector 2 - flatline duration (eps-relative, calibration-insensitive).
enable_flatline = true
flatline_max_jitter = 20      # × machine precision (np.finfo.eps)
flatline_max_duration_s = 5.0

# Detector 3 - amplitude outlier (robust z of per-channel σ_M).
enable_deviation = true
deviation_z_threshold = 5.0

# Detector 4 - windowed cross-channel correlation.
enable_correlation = false
correlation_threshold = 0.3
correlation_window_seconds = 1.0
bad_window_fraction_threshold = 0.05

# Detector 5 - HF / broadband σ_M ratio (one-sided positive z).
enable_hf_noise = true
hf_noise_z_threshold = 5.0
hf_lower_hz = 50.0
```

E.3 Epoch rejection

`[Mark detected artifacts]` owns no detector but does two things §3.12: it **conditions** the upstream marks into the final spans every consumer reads (always), and optionally **drops** segments. The `[segment_rejection]` block carries the span-conditioning knobs, the per-mode "Drop marked" toggle, and the drop policy (the two knobs that turn the conditioned marks into a drop):

Parameter	Default	Description
`auto_reject`	`true`	Apply automatic segment rejection (the event-mode **"Drop marked"** checkbox). Checked → drop the marked segments; unchecked → mark-only (cut/save/stats run, nothing drops).
`artifact_threshold`	`0.3`	Per-channel contaminated-sample fraction (0–1) that drops a segment (lower = stricter).
`artifact_min_run_ms`	`200.0`	G1 run-length trigger (ms), OR-ed with the fraction rule. `0` = off.

The per-segment artifact rule that decides *which* segments are dropped (fraction `artifact_threshold` or longest-run `artifact_min_run_ms`, both in `[segment_rejection]`, §3.12) is where you tune drop sensitivity. The epoch cut itself uses `reject=None`—there is no amplitude threshold at the cut; contamination is scored entirely from the upstream artifact marks (§3.11).

E.4 ICA / EOG (advanced)

Inside `[ICA_EOG]`. See §3.9 ICA for the full table; the parameters below are the ones not normally set in a standard config.

Parameter	Default	Description
`decim`	`1`	Subsampling factor for the fit. Only safe when the fit copy is band-limited (`fit_highpass_hz` and/or `fit_lowpass_hz`); without a low-pass leave at 1.
`random_state`	`42`	RNG seed.
`iclabel_min_prob`	`0.0`	Minimum classifier confidence for the top-1 label—a probability from `0` to `1` (`0.0` = no minimum). Below this threshold the component falls into `other`.

E.5 Parallelism (advanced)

For the basic field on the Config tab and what `n_jobs` accelerates, see Parallel jobs.

How to choose a value

The GUI shows you everything you need to decide, so you don't have to guess. After typing a number in the **Parallel jobs** field, three lines appear below it:

1. **Primary label**—`→ sequential (no parallelism)` when the field is `1`, or `→ N parallel workers` when larger.
2. **System info line**—detected hardware, e.g. `8 physical / 16 logical cores, 16 GB RAM, each worker ≈ 250 MB baseline + working memory`. Always shown. Physical core count is probed via `psutil` if installed, otherwise via `sysctl` (macOS), `/proc/cpuinfo` (Linux), or `ctypes` (Windows), with logical-cores-divided-by-two as a conservative fallback.
3. **Warning line** (yellow, only when your value is risky). Multiple warnings can stack:
   - more than half of physical cores → GUI can stutter while workers are busy;
   - more than physical cores → hyperthreads share execution units, diminishing returns;
   - more than logical cores → pure contention, strictly worse than capping at logical;
   - baseline RAM footprint (`n_jobs × ~250 MB`) exceeds one third of detected RAM → risk of swap thrashing.

Why BLAS is pinned to one thread per worker

IDE4EEG installs a global `joblib.parallel_config(backend="loky", inner_max_num_threads=1)` once per pipeline run. The `inner_max_num_threads=1` setting forces NumPy's BLAS backend (OpenBLAS / MKL / Accelerate) to use a single thread inside each worker. Without this pin, `N` joblib workers on a machine where BLAS defaults to `cpu_count` threads would create `N × cpu_count` OS threads, exhausting CPUs and triggering OS-level scheduler pathologies. The pin guarantees that the total CPU load equals `n_jobs`—exactly what the warning line shows. It also makes bit-exact reproducibility of floating-point sums possible regardless of `n_jobs`, because BLAS summation order is deterministic with a single thread.

Concretely: setting `n_jobs = 4` on a 16-thread BLAS will use four CPU cores, not 64.

Packaged desktop builds clamp catalog analyses to one worker

When IDE4EEG runs as a frozen standalone build (the Windows / macOS installers and portable bundles), the MNE catalog analyses are forced to `n_jobs = 1` no matter what the Parallel jobs field says. The reason is structural: joblib's `loky` backend spawns workers by re-launching the Python interpreter, but a frozen build has no interpreter—it re-launches its own application executable. macOS tolerates this via `multiprocessing.freeze_support()`, but on Windows the worker-argument handoff fails (`not enough values to unpack` in `_freeze_support`), so the app crashes or hangs. The clamp lives in `ide4eeg/utils/parallel.py` (`mne_n_jobs`, which `mne_catalog` re-exports as `_n_jobs` for back-compat) and only triggers when `sys.frozen` is set, so it is invisible to a `pip` / conda install, which keeps full parallelism. This is the same mitigation the ICA step has carried for the same reason. A complementary global clamp in `main._configure_parallelism` sets the joblib default to 1 on frozen builds, so every *bare* `joblib.Parallel()` that inherits the global config—including connectivity bootstraps / short-time and dipole-fitting—is also single-process on frozen builds. MP decomposition is unaffected (empi is a standalone C++ subprocess and was never at risk).

MP decomposition inherits this value

The `empi` Matching Pursuit binary has its own `--cpu-workers` switch (`[matching_pursuit] → cpu_workers` in TOML, "use [ ] / N cores" field in the Preprocess tab). The **default is to inherit from** `parallelism.n_jobs`—the Preprocess field shows the inherited value as grey italic placeholder text and refreshes live whenever you edit the Config tab field.

You only need to set `matching_pursuit.cpu_workers` explicitly if empi's memory model lets you go higher than joblib allows: empi workers share memory inside a single C++ process (~10–50 MB each), while joblib loky workers are full Python interpreters (~250 MB each). On a RAM-constrained machine the two can reasonably diverge; the explicit override exists for exactly that case. Otherwise leave it blank.

E.6 Hash-based caching internals

IDE4EEG's hash-based caching (Hash-based caching (overview)) uses a Merkle-style chain of per-step hashes: each step's hash covers its own config subset plus the hash of every preceding step that influenced its input. The per-step chain is seeded with the **signal-setup hash**—a digest of the fixed channel-selection + montage

config (`choosing_channels`, `electrodes_layout`), which always runs before the reorderable steps—so editing the kept-channel subset or the electrode layout invalidates every downstream snapshot. The seed also folds in the **input file's identity**: its basename, byte size, and modification time. That is deliberately *metadata*, not a content digest—one `stat` call, because EEG recordings run to gigabytes. Without it, snapshot reuse would key on the configuration alone, so re-recording or replacing a file *under the same name* with the config unchanged would silently reuse the previous run's snapshot over the new data. The MP-book hash (`_compute_mp_book_hash`) folds in the same three fields, so the atom book and the per-step chain go stale together. An input file that is missing or cannot be read is simply omitted from the seed rather than recorded as absent, so a hash pinned in the GUI before the file is available still agrees with the one the runner computes later. When a saved snapshot's stamped `[cfg:<hash>]` matches the current config's hash, the snapshot is reused; otherwise it's regenerated.

What's included in / excluded from each step's hash.

- **Included:** the step's own TOML section (e.g. `[filters]` for the filtering step), `preprocessing.step_order` (so reordering invalidates), and the recursive hashes of all preceding steps.
- **Excluded** (universal): GUI runtime fields (`_input_dir`, `_input_file_name`, etc.), output paths, log levels, the `parallelism` block (parallelism doesn't affect results, only speed), tool paths, and `allow_manual_modifications`—the manual-review master switch is a pure GUI affordance with no pipeline effect, so it is in no hash: toggling it never invalidates a step snapshot or the MP book. (The legacy `manual_*` decision keys can't affect a hash either—they are stripped from the config before it is hashed.)
- **Step-specific exclusions:** the standalone MP book hash strips `_MP_CFG_RUNTIME_FIELDS` and the `[matching_pursuit.filter]` sub-block (the latter a legacy, unused sub-block nothing in `analysis/` consumes). The MP filter step has an analogous hash on `[mp_filter]` that strips its own `filter` + `save` sub-blocks; editing reconstruction criteria never invalidates either book.

Three invariants keep the chain stable:

1. **Hash never includes mutated config.** Side-effecting steps (bad-channel detection appending to `choosing_channels.bad_channels`, ICA populating `ICA_EOG.bad_sources`) freeze their hash *before* the mutation. Otherwise the second run's hash wouldn't match the first run's stamp.
2. **No deepcopy of bound methods.** Manual-review hooks (`_review_bad_channels`, `_review_ica_components`, `_review_bad_epochs`) are passed as closures, not bound methods. Deepcopying a bound method would try to deepcopy `self` (the QMainWindow), which fails on Qt's C++ state.
3. **Pin BEFORE truncation.** The GUI eye-click path force-pins the hash against the user-level pre-truncation cfg in `_run_truncated_pipeline`; the runner's preamble (`preprocessing.py`) and `_launch_pipeline` use `force=False` so an upstream pin survives. This guarantees that a re-entry click on the same eye sees the same hash the truncated run stamped.

These invariants are tested by `tests/test_view_step_result.py` and `tests/test_preamble_phase1.py`.

Appendix F. config.toml vs Export Script

Two ways to capture and re-run a configuration outside the GUI:

1. `config.toml`—declarative TOML, the same format the GUI uses internally. Save with **Save Config...**, run with `ide4eeg --run myconfig.toml`.
2. **Exported Python script**—a self-contained `.py` file calling `ide4eeg.api.run_file()` with the current parameters. Generate with **Export Script...** (Config tab or **File → Export Script...**), run with `python3 generated_script.py`.

Both share the same code path under the hood—same preprocessing pipeline, same analysis modules, matching results (up to small floating-point differences under parallelism).

F.1 Side-by-side

Aspect	`config.toml`	Exported script
Format	declarative TOML	imperative Python
Round-trips with the GUI	yes (**Load Config...**)	one-way export only

Aspect	`config.toml`	Exported script
Verbosity	compact (only the keys you set)	one keyword arg per non-default value
Loops over files / subjects	built-in folder batch (`--run` a directory); per-file configs need a shell loop	inline Python `for` loop (see F.5)
Conditional logic / parameter sweeps	needs templating	native Python branching
Custom callbacks (manual-review hooks)	not expressible	not expressible via `run_file` either—the `manual_hooks=` mechanism is internal to `main.main`, reached via the GUI
Reads as documentation	requires manual lookup of TOML keys	reads as a worked example of `ide4eeg.api`
Diffability across runs	excellent (line-by-line text diff)	OK (kwargs block diffs cleanly)
External invocation	`ide4eeg --run cfg.toml`	`python3 script.py`
IDE4EEG location	implicit (whatever's on `PYTHONPATH`)	explicit `sys.path.insert(0, "<repo>")` line at the top
Disabled features	every key persists in the file	feature-gated sub-configs are pruned (e.g. `[ICA_EOG]` is omitted when `prepare_ica = false`)

F.2 When to prefer `config.toml`

- Routine batch jobs run across many recordings via shell loops or cluster submission.
- Sharing with collaborators who use the GUI: they can **Load Config...** and edit interactively.
- Parameter sweeps via templating (e.g. one TOML per condition, generated by another script).
- Quick changes on disk without regenerating the file from the GUI.

F.3 When to prefer the exported script

- **Archival reproducibility.** Check the `.py` into the results folder alongside the outputs. The `sys.path.insert(...)` line records exactly which IDE4EEG checkout produced the results—no separate "what version was this?" question.
- **Custom Python wrapping.** Loops over subjects with conditional logic, post-processing in the same script, `pandas` / `matplotlib` follow-ups inline.
- **Teaching / API discovery.** The script makes the `ide4eeg.api.run_file()` surface explicit and is a good starting point for users learning the programmatic API.
- **Custom Python wrapping around `run_file`.** Note that per-file manual-review callbacks are *not* expressible through `run_file` (its signature is `run_file(input_path, output_path=None, overwrite=True, **config_overrides)`; the `manual_hooks=` mechanism is internal to `main.main` and reached via the GUI)—but you can still surround `run_file` with your own Python pre-/post-processing.

F.4 What both leave on disk

When you run a pipeline **from the GUI** (Run Pipeline, or a per-analysis Run with its in-flight narrowing), the config the GUI collected is written as `config.toml` in the output directory, so you can later **Load** it back into the GUI. The `--run` CLI and the exported-script route do not auto-write a `config.toml` to the output folder—save one from the GUI (or keep your source TOML) if you want that record.

F.5 Processing multiple files (batch)

Three ways to run the same pipeline over many recordings—all are the pure function `f(input, config)` mapped over each file, one file at a time (see Batch processing in §5 for the reproducibility framing).

Route 1—one config, a whole folder (`--run`). Point `input_path` at a *directory* and run headless:

```
ide4eeg --run myconfig.toml          # every supported file in the folder
```

IDE4EEG discovers each file, processes it independently with the *same* config, and writes its own `IDE4EEG_OUT_<file>/`; a failed file is logged and skipped, the batch continues. The GUI can't run a folder (**Run Pipeline** on a directory is refused as batch-only)—prepare the config in the GUI on one representative recording, **Save Config...**, set **Input** to the folder, then `--run` it. A batch is always fully automatic: manual review decisions are per-recording GUI-session state and are never in the config, and any legacy `manual_*` key in a hand-edited config is stripped at load and ignored (see Batch is automatic-only).

Route 2—exported script, uncomment the loop (pure Python). Export Script... writes a self-contained `.py` with the config inlined (no TOML) calling `run_file()` on one file. It ends with a ready-made, commented **"process multiple files"** block—uncomment it and set the glob:

```
import glob, pandas as pd

all_stats = []
for filepath in glob.glob("data/subject_*.vhdr"):
    r = run_file(
        filepath,
        output_path="output/",
        resample_freq=256,                # ← the SAME inlined config
        filters={"highpass_freq": 0.5, "lowpass_freq": 30.0},
        segmentation={"mode": "epochs"},
    )
    r.data["subject"] = filepath
    all_stats.append(r.data)

pd.concat(all_stats).to_csv("group_statistics.csv")
```

The generator pre-fills that loop's keyword arguments with your exported config, so uncommenting applies the identical recipe to each file and aggregates every file's per-segment statistics into one CSV. (Exported with the **portable** option, paths are wrapped in `_here()` / `_inhere()` anchors so script + data move together—glob real paths and pass a plain `output_path="output/"`.)

Route 3—hand-written `run_file` loop (per-file config). `run_file()` is single-file, so *you* own the loop—which means you can give each file a *different automatic* config (different resample rate, filter band, reference, ...) and add conditional logic the single-config folder batch can't express:

```
from ide4eeg.api import run_file
import glob, os

base = dict(resample_freq=256,
            filters={"highpass_freq": 0.5, "lowpass_freq": 30.0})
per_file = {"s01.vhdr": {"resample_freq": 512},   # this one was recorded faster
            "s02.vhdr": {"reference": {"mode": "average"}}}

for f in glob.glob("data/*.vhdr"):
    run_file(f, output_path="out/", **base,
             **per_file.get(os.path.basename(f), {}))
```

Each call is a pure single-file `f(input, config)` with *its own* config. Note this is **automatic only**: manual review decisions (bad channels, ICA components, epoch/gaze rejection) are a GUI interaction—they are not config values and the `manual_*` keys are stripped at load, so they cannot be scripted here. For a channel known-bad in specific files, use `choosing_channels.dropped_channels` (a real config field).

Which route.

You want...	Route
the same automatic recipe over a folder, headless	1 (`--run`) or 2 (exported loop)
archival reproducibility + inline `pandas` / `matplotlib` follow-ups	2 (exported script)
a *different* automatic config per file, conditional logic	2 or 3 (you own the loop)

Appendix G. Pipeline invariants (rule catalogue)

Read-only list of the consistency rules IDE4EEG checks before it runs the pipeline (see the *Consistency rules* section). This appendix is generated automatically from the rule registry when the PDF is built, so it always matches the installed version. In the live application the same list appears at the bottom of the **Help** tab, where you can additionally dismiss a rule or restore a dismissed one — those actions exist only in the GUI.

Hard rules (block the pipeline run)

Reference & re-referencing

- `hard.reference.channels_present` *(fires at: signal_load, field_edit, pre_launch)* — Named re-reference channels must exist in the signal; otherwise the pipeline's _set_reference silently drops them and the recording is left at its native reference.
- `hard.reference.car_blocks_inverse_modeling` *(fires at: field_edit, pre_launch)* — MNE inverse modeling (dSPM / LCMV / MxNE) computes a forward model against fsaverage assuming an unreferenced (or projector-referenced) EEG. A subtractive re-reference like CAR shifts the data into a basis the forward model can't invert, and MNE refuses with 'Custom EEG reference is not allowed for inverse modeling'. Without this rule the conflict surfaces only at analysis time, as a per-condition ERROR log inside the MNE catalog dispatcher's try/except (which then reports 'N/N analyses completed' with no indication that 6 of them produced no output) — a silent partial-success that wastes the full preprocessing pass. Surface the conflict at preflight instead.
- `hard.reference.rest_requires_positions` *(fires at: signal_load, field_edit, pre_launch)* — REST re-referencing (Yao 2001) builds a sphere head model + volume source space + forward solution, so every EEG channel needs an electrode position. If the file has no native positions AND no montage step runs before the reference step, _set_reference raises 'REST re-reference needs electrode positions' mid-pipeline. This is the REST sibling of hard.step_order.iclabel_requires_positions — caught pre-launch so the user fixes the montage step before the run reaches the reference step.

Dipole fitting

- `hard.dipole.ref_channel_present` *(fires at: signal_load, field_edit, pre_launch)* — dipole_fitting.ref_channel names a single channel (or 'average') used as the reference for MNE's fit_dipole. If the named channel isn't in the signal, MNE's set_eeg_reference silently leaves the reference unchanged and the dipole fit runs against the raw reference – a known silent-failure mode.

Channel selection

- `hard.channels.ica_eog_channels_present` *(fires at: signal_load, field_edit, pre_launch)* — ICA's find_bads_eog selector references EOG channels by name; missing names are dropped with a log warning, and ICA falls back to whichever channels remain.
- `hard.channels.dipole_ignored_channels_present` *(fires at: signal_load, field_edit, pre_launch)* — dipole_fitting.ignored_channels lists channels to exclude from the fit. Misnaming an ignored channel means it's NOT excluded – silently weakening the fit.
- `hard.channels.eeg_profiles_channel_present` *(fires at: signal_load, field_edit, pre_launch)* — eeg_profiles.channel picks ONE channel from the MP book. A string value must match a channel name; an unmatched one ABORTS the analysis (eeg_profiles.py raises ValueError – there is no fallback for the string form). Int values are book indices and are range-checked at run time instead.
- `hard.channels.matching_pursuit_channels_present` *(fires at: signal_load, field_edit, pre_launch)* — MP decomposition writes a book whose channels are matching_pursuit.channels. Missing names are dropped, shrinking the book to whatever's available – consumers (dipole, eeg_profiles, mp_filter) then can't find what they expected.
- `hard.channels.connectivity_channels_present` *(fires at: signal_load, field_edit, pre_launch)* — connectivity.channels selects channels for the MVAR fit. Missing channels are silently dropped; the user sees a smaller-than-expected connectivity matrix.
- `hard.channels.choosing_channels_present_subset` *(fires at: signal_load, field_edit, pre_launch)* — choosing_channels carries the user's manual subset (selected or dropped). Stale TOMLs from a different recording can name channels that don't exist; the pipeline silently filters those out and proceeds with whatever matched.

Segmentation modes & events

- `hard.modes.event_locked_mode_no_stim_channel` *(fires at: signal_load, field_edit, pre_launch)* — Event-locked mode needs event markers. They may sit in a STIM channel OR in annotations / tags — the file readers fold annotation/tag events into a synthetic STIM channel at load (input.py::_append_synthetic_stim), so the real precondition is 'markers exist somewhere', not 'a stim-typed channel is already present'. Only when there are no markers at all does find_events cut zero epochs – caught early so the user can switch modes.
- `hard.modes.event_locked_mode_no_events_selected` *(fires at: field_edit, pre_launch)* — Event-locked mode needs at least one event picked in the Events panel. An empty selection produces zero epochs at cut_segments time.
- `hard.modes.event_locked_mode_events_not_in_trim` *(fires at: signal_load, field_edit, pre_launch)* — Selected event names must appear among the events the reader discovered. A stale config can carry tag names from a different recording; the pipeline would later raise 'zero epochs' with no actionable context.
- `hard.modes.eeg_profiles_with_event_locked_mode` *(fires at: field_edit, pre_launch)* — EEG profiles assumes a continuous time axis. With event-locked epochs the profile chart's x-axis covers only the contiguous epoch duration; detections in gaps between events fall off the canvas.

Preprocessing step order

- `hard.step_order.soft_ordering_constraints` *(fires at: field_edit, pre_launch)* — Recommended step orderings (bad-channel detection before ICA, and re-referencing before filtering). Reordering them won't crash the pipeline, but the literature and internal benchmarks show degraded downstream results.
- `hard.step_order.dipole_requires_mmp1` *(fires at: field_edit, pre_launch)* — Dipole fitting reads multivariate atoms (MMP1). Single-channel MP (SMP) atoms have no shared topography across channels and cannot be source-localised.
- `hard.step_order.mmp_requires_multichannel` *(fires at: field_edit, signal_load, pre_launch)* — MMP1/MMP3 are MULTICHANNEL matching pursuit — they fit atoms shared across channels. On a single channel that is degenerate and empi exits non-zero. Use SMP for single-channel data.
- `hard.step_order.dipole_requires_mp_available` *(fires at: field_edit, pre_launch)* — Dipole fitting consumes atoms from an MP book. Either MP decomposition must run in this pipeline or an existing book must be selected on the Analysis tab.
- `hard.step_order.eeg_profiles_requires_mp_available` *(fires at: field_edit, pre_launch)* — EEG profiles classifies atoms from an MP book. Either MP decomposition must run in this pipeline or an existing book must be selected on the Analysis tab.
- `hard.step_order.pinned_mp_book_exists_on_disk` *(fires at: field_edit, signal_load, pre_launch)* — When _mp_book_path is pinned to an existing book (typed in a prior session, picked on the Analysis tab) the file must still be on disk – otherwise the next pipeline run fails mid-step. Triggered by the pin itself and by every MP consumer's enable flag.
- `hard.step_order.mp_book_file_exists_on_disk` *(fires at: field_edit, signal_load, pre_launch)* — When an analysis is set to consume an explicit book file (mp_book_source = 'file'), that .db must exist on disk — otherwise the run fails when the analysis tries to read it.
- `hard.step_order.iclabel_requires_positions` *(fires at: field_edit, signal_load, pre_launch)* — The ICLabel ICA-component selector (and 'both', which runs ICLabel) classifies components partly from their scalp topography, so every EEG channel fed to ICA must carry an electrode position. If the file has no native positions AND no montage step runs before ICA, ICLabel aborts AFTER the (expensive) ICA fit with a terse 'Channel position is missing' error. Caught here pre-launch so the user fixes the montage step before paying for the fit.
- `hard.step_order.eeg_profiles_reject_needs_eeg_artifacts` *(fires at: pre_launch)* — EEG-profile 'reject artifacts' drops atoms whose time-centre lands in a BAD_* span carried ON the analysed signal. Those marks come from the 'eeg_artifacts' preprocessing step; without it in step_order the signal has no carrier and reject_artifacts is a SILENT no-op (atoms in the gray band are still counted), even though an eeg_artifacts tag may exist on disk and be drawn as the band. Warn before the run (2026-06-22 incident).
- `hard.step_order.eeg_artifacts_has_no_detector` *(fires at: field_edit, pre_launch)* — The 'eeg_artifacts' step is a container for four independent detectors (p2p / slope / muscle / flat). With all four off the step still runs and still writes its -tag.txt, but the file is EMPTY and nothing is ever marked — an inert step that looks active in the step list. The GUI used to silently re-arm all four instead; a config is a statement of intent, so warn and obey (C-CFG-6, 2026-07-20).

External tool availability

- `hard.tools.mp_decomposition_requires_empi` *(fires at: field_edit, pre_launch)* — MP decomposition uses the empi binary to build the atom book. Without it on this machine the step fails immediately with 'empi: command not found' or a Java ClassNotFoundException. Reusing an existing book (when _mp_book_path is pinned) bypasses empi. MP filter also calls empi via its internal decomposition, so it carries the same requirement.
- `hard.tools.connectivity_requires_connectivis` *(fires at: field_edit, pre_launch)* — Connectivity analysis runs without ConnectiVIS, but the 3D source-space viewer (cvis-gui) won't be reachable from the Output tab. The numeric CSV outputs are unaffected, hence warning not error.
- `hard.tools.iclabel_requires_backend` *(fires at: field_edit, pre_launch)* — The ICLabel ICA-component selector (and 'both', which runs ICLabel) needs mne-icalabel's neural-net backend – onnxruntime or pytorch. Neither is a core dep (onnxruntime has no wheel on the newest CPython), so a config requesting iclabel on a backend-free machine would fail mid-run. The GUI grays out the choice, but a hand-written / ported TOML can still carry it – this rule catches it before launch.
- `hard.tools.readmanager_version_outdated` *(fires at: pre_launch)* — IDE4EEG pins readmanager>=2.0.0 (pyproject). 2.0.0 makes the read/write proxies GC-safe (no leaked *.obci.raw file handles); and >=1.6.3 gives chtype_heuristic its respiration rule (Resp/SaO2/thermistor -> 'resp') – an older readmanager leaks handles and silently classifies those channels as 'bio'/'misc', so PSG montages load with the wrong types and no error (the 2026-06-08 Windows report). An environment that kept a stale wheel is caught here at launch. Warning not error: pure-EEG runs on a fresh install are unaffected.
- `hard.tools.gaze_requires_reference_dir` *(fires at: field_edit, pre_launch)* — Gaze (not-looking) detection scores each video frame against reference faces. With no video_reference_dir set (or a missing one) the gaze step can't run: it logs one WARNING and skips, producing zero not-looking marks while the pipeline still reports success – so an enabled Detect-gaze step silently does nothing. Surfacing it at pre-launch turns that buried WARNING into a visible preflight notice. Warning, not error: the rest of the pipeline is unaffected.

Numeric Ranges

- `hard.numeric_ranges.filter_hp_lp_ordering` *(fires at: field_edit, pre_launch)* — When highpass_freq > lowpass_freq the IIR/FIR design produces an empty passband – the filter silently zeros the signal across that range instead of raising. Easy to hit by swapping the two values during config editing.
- `hard.numeric_ranges.filter_above_nyquist` *(fires at: signal_load, field_edit, pre_launch)* — Filter cutoff at or above sfreq/2 (Nyquist) leaves IIR designs unstable and FIR designs with wraparound artefacts. Signal-loaded rule – needs the actual sampling rate to flag. Nyquist is computed against the EFFECTIVE rate at the filter step: when an enabled resample precedes filtering, the filter runs at the resample target, not the original recording rate.
- `hard.numeric_ranges.dipole_min_gof_range` *(fires at: field_edit, pre_launch)* — dipole_fitting.min_gof is a goodness-of-fit threshold as a PERCENTAGE in [0, 100] – MNE's dipole GOF is 0-100 and min_gof is compared to it directly (analysis/dipole/fitting.py: gof >= min_gof). 0 keeps every dipole; 50 keeps only dipoles fitting >= 50 %. A value < 0 keeps everything; > 100 filters every dipole out.
- `hard.numeric_ranges.epochs_offset_ordering` *(fires at: field_edit, pre_launch)* — When start_offset >= stop_offset the epoch window has zero or negative duration. The pipeline silently resets to the default [-0.3, 0.7] with only a log warning (channels_and_signal.py:617-633). Surface it as an error so the user sees their values were discarded.
- `hard.numeric_ranges.mp_iterations_or_energy_required` *(fires at: field_edit, pre_launch)* — empi needs at least one stopping criterion – if both iterations and explained_energy are 0, the decomposition hangs. An explained_energy > 1.0 silently caps at 1.0 (no error, no log).
- `hard.numeric_ranges.connectivity_mvar_order_range` *(fires at: field_edit, pre_launch)* — connectivity.mvar_order=0 triggers AIC auto-selection (documented). Negative values are nonsensical; very large values trip a silent per-tag skip-with-log instead of erroring out.
- `hard.numeric_ranges.mne_catalog_freq_bounds` *(fires at: signal_load, field_edit, pre_launch)* — MNE catalog spectra (PSD multitaper/Welch/topo) and time-frequency (TFR Morlet/multitaper/ERD-S) entries take frequency-axis bounds via fmin/fmax (PSD) or freq_min/freq_max (TFR). A negative value slips past the truthy-zero-sentinel and renders an empty axis or draws from 0 silently — issue #24. A lower bound above Nyquist empties the freq axis and crashes the analysis (checked once sfreq is known) — E6.

- `hard.numeric_ranges.mp_freq_bounds` *(fires at: field_edit, pre_launch)* — Matching-pursuit atom-filter and Gabor-display upper bound silently drop all atoms (or render an empty spectrum) when freq_min < 0 or freq_max < 0 because the truthy-zero sentinel only catches exact 0.
- `hard.numeric_ranges.dipole_freq_bounds` *(fires at: field_edit, pre_launch)* — dipole_fitting.freq_min / freq_max filter atoms before the dipole fit. A negative value silently drops every candidate atom (truthy-but-impossible bound) and the pipeline reports 'no dipoles' with no diagnostic.
- `hard.numeric_ranges.eeg_profiles_freq_bounds` *(fires at: field_edit, pre_launch)* — EEG-profile structures (alpha / spindles / slow waves etc.) match atoms by freq_min..freq_max. A negative bound silently drops every atom for that structure.
- `hard.numeric_ranges.trim_end_within_duration` *(fires at: signal_load, field_edit, pre_launch)* — trim_end $>$ file duration was silently clamped to file end by channels_and_signal._trim_signal – the user thinks they trimmed (e.g. expecting 30 s of data from a 120 s file with trim_end=30) but the pipeline kept everything past trim_start. Surfaces the silent clamp as a warning so the user can fix the typo (or accept it knowingly).
- `hard.numeric_ranges.resample_below_sfreq` *(fires at: signal_load, field_edit, pre_launch)* — MNE's resample() happily up-samples when target $>$ source sfreq, but for EEG that's almost always unintended (a typo for 'downsample to 256' that drifted to 2560). Up-sampling adds no information; warn so the user can confirm the rate or fix it.
- `hard.numeric_ranges.montage_layout_known` *(fires at: pre_launch)* — A typo in [montage] layout (e.g. 'standard_1004' for 'standard_1005') previously aborted the pipeline mid-run with a generic ValueError that echoed neither the bad value nor the list of valid names. Pre-flight check against mne.channels.get_builtin_montages() lets the GUI badge the typo before run. pre_launch only (not field_edit) – the lookup is cached but the first call costs ~1.4 s which would freeze typing the first time.
- `hard.numeric_ranges.ica_n_components_within_rank` *(fires at: signal_load, field_edit, pre_launch)* — ICA_EOG.n_components is passed straight to MNE's ICA(n_components=…) with no clamping (_resolve_n_components in preprocessing/ica.py). An integer above the data rank — at most the EEG channel count, one fewer with average reference — crashes the fit at run time instead of being capped; a value < 1 is likewise rejected by MNE. 'rank' (the default) and a float in (0, 1) are always safe, so only integers are checked. Surfaced pre-run so the user sees it before the fit aborts.