TROUBLESHOOTING & OPS

OpenSAI FAQ, Support & Maintenance

Quickly diagnose setup, connectivity, application dependency, and adaptation behavior issues, and keep your OpenSAI deployment healthy over time.

Overview | Quickstart | User Manual | Agent Integration

Keep the original `job_id` handy whenever you investigate a run so you can correlate CLI output, Web console views, and `.run/` process logs.

Quick Symptom Triage

Symptom	Typical Cause	First Action
`opensai: command not found`	Project virtual environment `.venv` is not active	Activate `.venv` (`source .venv/bin/activate` or `.venv\Scripts\Activate.ps1`) and run `opensai --help`
Console opens, but your local worker is missing	Simulated **Sample data** mode is active, or worker is not connected	Click `Switch to live data` in the top banner (or disable sample data under **Settings**), then run `opensai devices list`
Application or DAG job ends in `FAILED`	Missing Python library on worker, missing dataset, or step exception	Open the failed run and step under **Runs** in the console to read its exact error message
Dataset files exist on disk, but the app cannot find them	Dataset is not a direct child of `OPENSAI_DATA_DIR`, or name differs	Run `opensai data list` and match the reported dataset name to `app.yaml`
Wait command times out, or MCP returns `done=false`	Job exceeds the client wait window and is still executing	**Do not resubmit**; query the existing `job_id` via `opensai result get YOUR_JOB_ID`
Knobs stay unchanged even with a deadline set	Available resources already meet the deadline at default quality, or `-k` pinned the knob	Inspect candidates with `opensai explain YOUR_JOB_ID`; enable `adaptive` mode for joint placement + knob tuning

1. Installation & First Run

Do I need multiple computers or a dedicated GPU to get started?

No. A single laptop can run the coordinator hub, a worker process, and the Web console. `pi_pipeline`, the starter template, `number_stats`, `cluster_analytics`, and lightweight `coop_perception` all run on CPU. You only need a GPU and PyTorch on workers that execute deep-learning steps such as `how2comm_perception`.

Are Node.js, Docker, or Kubernetes required?

No. The release ships with a prebuilt Web console, and tasks execute directly inside lightweight Python worker pools with OS-level core affinity and resource limits. Node.js is only needed if you modify `frontend/` source code and rebuild the UI bundle.

What if Python package downloads are slow?

Point `PIP_INDEX_URL` to a fast regional PyPI mirror before running setup:

```bash
PIP_INDEX_URL=https://pypi.tuna.tsinghua.edu.cn/simple bash scripts/setup.sh
```

Core functionality requires only lightweight base dependencies—install ML packages later only on workers that need them.

2. Web Console & Multi-Network Connectivity

The console loads, but the sidebar says "Coordinator offline" or "Disconnected"

The browser UI is served by `user_client.py` (bound to `127.0.0.1:5000` by default), which maintains an auto-reconnecting link to the coordinator hub (`TCP 8888`). Run `opensai cluster status` in a terminal to isolate the issue: if the CLI also fails, check that `server.py` is running on the expected address; if the CLI succeeds, verify **Bridge address** under **Settings** in the console (leave empty when served from the bridge itself).

Why can't another machine open `:5000`, and why are "System parameters" read-only?

- **Loopback-only default**: Because the Web bridge has no built-in authentication, it binds to `127.0.0.1:5000` by default. On a trusted LAN, set `OPENSAI_BRIDGE_HOST=0.0.0.0` before starting the bridge, or forward the port safely over SSH (`ssh -L 5000:127.0.0.1:5000 <hub-host>`).
- **Startup environment variables**: Policies such as `OPENSAI_PLACEMENT` and `OPENSAI_MULTIPATH` are read when the hub or worker starts—set them in the environment and restart the process; the **System parameters** table under **Settings** documents active defaults.

How do I connect devices across separate LANs?

Establish virtual private connectivity among the hub, clients, and workers first using Tailscale, WireGuard, or a corporate VPN, then set `OPENSAI_HOST` to the hub's reachable private IP.

> ▲ **Warning**
>
> **Ports & network boundary**: The hub control plane (`TCP 8888`) has no built-in public authentication and must only be exposed inside trusted LANs or VPNs. In addition, cross-device jobs require peer-to-peer reachability on each worker's QUIC data-plane port (`UDP 9999` by default)—hub reachability alone is not enough to transfer intermediate artifacts between workers.

3. Applications, Dependencies, Data & Models

Registration succeeded—why did a step fail with `ModuleNotFoundError`?

`opensai app register` distributes `steps/*.py` source files to workers automatically, but does not install third-party packages remotely. Check `requires` in `app.yaml` and install those packages inside `.venv` on every worker eligible to run the step.

Where should model weights and large datasets be stored?

Store them on the workers that execute the corresponding loading or inference steps:

- **Datasets**: Place each dataset as an immediate child file or folder under `Offloading-System/data` (or `OPENSAI_DATA_DIR`), or symlink external storage there, and verify visibility via `opensai data list`.
- **Model weights**: Weights are not bundled into code synchronization; place them at the path expected by the application on GPU inference nodes.

Does `video_counting` use a live camera? Can I plug in my own YOLO model?

`video_counting` generates synthetic frames with `numpy` against known object counts so you can test dual-branch pipelines and resolution/stride knobs anywhere. To process live RTSP streams, video files, or a custom YOLO detector, replace the capture and detection steps, install OpenCV / PyTorch and weights on eligible workers, expose tunable resolution or frame stride as `knobs`, and keep business-critical thresholds fixed.

How can a job report `SUCCEEDED` when some steps are `SKIPPED`?

Applications can mark transient collaborators (such as an assisting vehicle leaving an intersection) with `optional: true`. If that participant is offline, lacks data, or returns `SKIP:`, OpenSAI skips the optional branch cleanly so required downstream fusion steps still finish on time using remaining inputs.

4. Understanding Metrics & Adaptation

Does setting `--deadline` guarantee completion within that time?

A deadline guides device placement, core allocation, and knob selection—it cannot override physical hardware limits. Check `duration_s` and `qos` attainment in the job result: if even the lightest knob tier misses the target consistently, add compute capacity, enable GPU execution, or lower the input rate.

Why isn't multi-device execution always faster, and why might multipath still use a single direct link?

- **Parallelism vs. transfer cost**: Small tasks, strictly sequential chains, or steps producing huge intermediate payloads may run faster co-located on one machine. Compare `compute_s` and `fetch_s` per node to see where time goes.
- **Selective multipath**: With `OPENSAI_MULTIPATH=1`, the `DELTA` planner evaluates payload size, setup overhead, spare relay capacity, and shared bottlenecks per transfer. Small artifacts or paths sharing a single saturated NIC rightly stay on direct QUIC.

Why doesn't `duration_s` equal `compute_s + fetch_s + queue_s`? Does quality `0.9` mean 90% accuracy?

- `duration_s` is end-to-end wall-clock job time, whereas `compute_s`, `fetch_s`, and `queue_s` sum across all nodes; overlapping parallel branches can make summed node times exceed wall time.
- Quality scores (`0` – `1`) follow each application's declared metric (for example, relative F1, sample coverage, or normalized accuracy). Always compare configurations using the same application quality definition.

5. MCP & Agent Skill

I imported `SKILL.md`, so why don't any `opensai_*` tools show up?

`SKILL.md` provides agent instructions, while callable tools come from the `opensai_mcp` server. Follow Sections 1–2 of the Agent Integration Guide to install `--agent` dependencies, configure the MCP server, and verify connectivity with `python -m opensai_mcp --check`.

6. Routine Maintenance, Archiving & Upgrades

In-memory job records and intermediate worker artifacts expire automatically (`OPENSAI_JOB_TTL_S`, `OPENSAI_ARTIFACT_TTL_S`) and are not intended for permanent storage. Preserve the following across updates:

Asset	Recommended Practice
Custom application packages	Track `app.yaml`, `steps/`, and version tags in Git
Datasets and model weights	Keep on dedicated storage outside the release folder and mount via `OPENSAI_DATA_DIR` or symlinks
Startup environment & tokens	Document custom `OPENSAI_*` settings; store `OPENSAI_MCP_TOKEN` and `OPENSAI_ADMIN_TOKEN` securely
Important results & benchmarks	Export `result.to_dict()` as JSON via the SDK or write outputs to persistent storage
Diagnostic logs	Inspect `server`, `device`, and `user` logs under `.run/` when troubleshooting

Before upgrading, let active jobs finish, back up custom apps and settings, extract the new release into a clean directory, run setup, reconnect datasets, and validate with a quick smoke job before switching production callers.

7. Getting Help & Reporting Issues

Use Issues on github.com/INSAI-institute/OpenSAI for bug reports and Discussions for architecture questions or deployment scenarios. Useful reports include:

1. OpenSAI software and materials versions, OS, and Python version;
2. Cluster setup (single-host vs. multi-host worker count) and launch flags;
3. Minimal reproducible DAG or application manifest, full error output, and `job_id` (redact Bearer tokens, credentials, and private data before sharing logs).

Report credential leaks or security vulnerabilities privately via repository Security Advisories rather than public Issues (see Contributing & Citation; 中文常见问题与维护).