English Version →

软件使用与开发手册

OpenSAI 软件使用手册

部署边缘异构集群，封装与运行分布式应用，读懂每一次调度与自适应决策。

产品与技术概览 | 10 分钟快速上手 | 智能体接入指南 | FAQ 与排障维护

本手册面向使用 OpenSAI 的应用开发者、集群使用者与部署维护人员。初次接触建议先跟随 快速上手指南 完成 10 分钟本机闭环，日常开发与运维可直接按需查阅对应章节。若无特殊说明，文中所有命令均以克隆仓库（github.com/INSAI-institute/OpenSAI）或解压发行代码包后的项目根目录为当前工作目录，并在激活的项目 `.venv` 虚拟环境中执行。

面向应用开发者

通过声明式 `app.yaml` 清单与 Python 步骤类封装多阶段业务，定义数据亲和、可选分支与可调参数旋钮。

面向集群与实验使用者

在单机或多机异构集群上使用截止时间、最快完成或资源高效模式运行任务，直观对照耗时、质量与决策依据。

面向运维与智能体集成

规划控制面与 QUIC 数据面端口，按需开启中继多链路传输，并通过 MCP 与使用 Skill 安全开放给 AI 智能体。

研发与发布机构：复旦大学（Fudan University）· Institute on Networking Systems of AI (INSAI)
等价交互入口：双语 Web 控制台（`:5000`） · `opensai` CLI · `OpenSAIClient` Python SDK · `opensai_mcp` 智能体服务

目录

按任务目标快速查阅 03
1. 认识 OpenSAI 03
2. 安装与本地运行 05
3. 部署多机集群 06
4. 使用双语 Web 控制台 08
5. 运行和管理应用 09
6. 编写可复用应用 11
7. 直接编排与提交 DAG 14
8. 选择时间、质量与资源目标 15
9. 数据管理与多链路传输 16
10. 查看结果和性能 17
11. Python SDK 17
12. CLI 速查 18
13. 智能体接入 19
14. 常用运行配置速查 19
15. 维护与帮助 20

按任务目标快速查阅

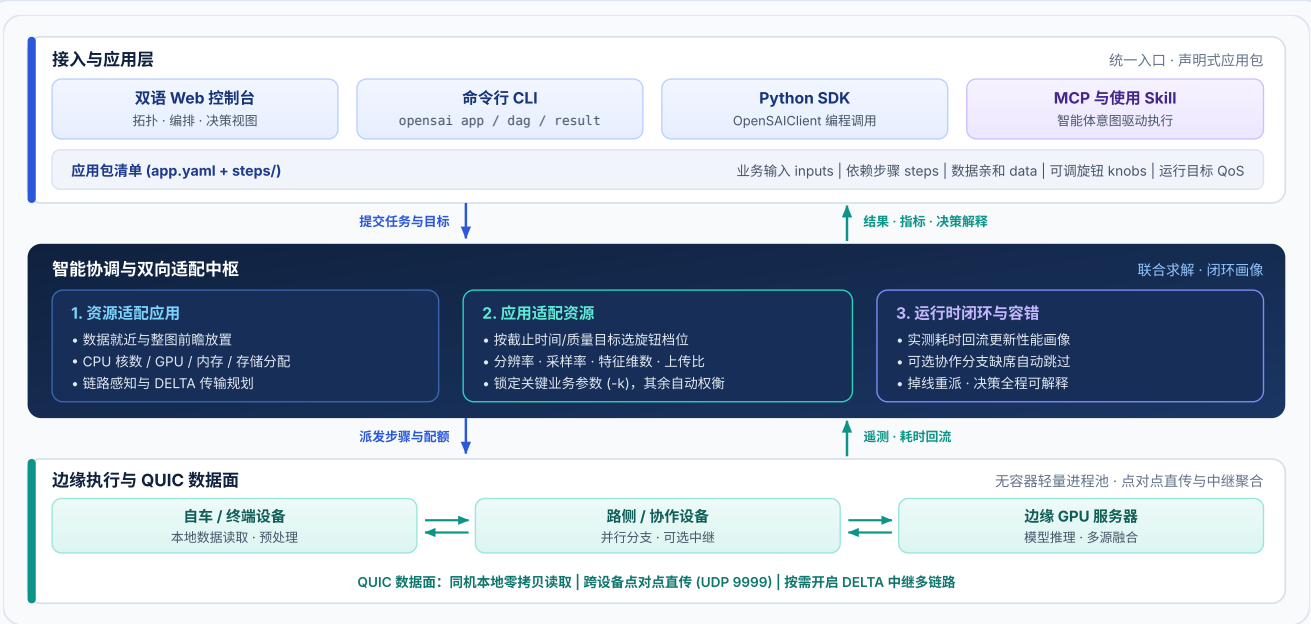
你当前要完成的事	推荐直达入口
首次在一台电脑安装并跑通第一个任务	快速上手指南 · 第 2 章：安装与本地运行
将多台电脑 / 边缘服务器组建成集群	第 3 章：部署多机集群（端口规划与角色标签）
在可视化界面编排任务、查看拓扑与决策	第 4 章：使用双语 Web 控制台
运行内置感知/聚类示例，或管理应用版本	第 5 章：运行与管理应用（含协同感知与 How2comm 指南）
将自己的 Python / 模型代码封装为应用包	第 6 章：开发可复用应用包
临时组合已有算法并直接提交 JSON DAG	第 7 章：直接编排与提交 DAG
配置截止时间、质量底线与联合放置模式	第 8 章：选择时间、质量与资源目标
管理本地数据集，开启 QUIC 中继多链路	第 9 章：数据管理与多链路传输
解读耗时、核秒指标、决策依据与 Python/CLI	第 10 章：查看结果和性能 · 第 11 章：Python SDK · 第 12 章：CLI 速查

1. 认识 OpenSAI

1.1 平台定位与整体架构

OpenSAI 将多台分散的 CPU / GPU 边缘设备组织为一个统一可调用的分布式计算环境。开发者只需通过声明式清单描述应用步骤、输入数据亲和与运行目标，平台即可自动按依赖关系派发步骤、分配四维资源、跨设备传输中间产物，并将业务输出与完整决策解释一并返回。

- **对应用开发者**：专注编写单步算法与业务逻辑，无需手写跨机 Socket 通信、重试或负载均衡。
- **对集群使用者**：用同一套 Web、CLI、SDK 或智能体接口调用不同领域应用，按需切换时延或质量目标。
- **对运维维护者**：实时掌握各节点核心、内存、出口带宽与存储水位，逐笔回溯放置与调参依据。



OpenSAI 三层系统架构与双向适配闭环

1.2 五个核心概念

概念	在 OpenSAI 中的准确含义
中枢（Hub / Coordinator）	接收客户端请求、维护设备与算法账本、执行联合调度决策并汇总结果的中心协调进程（`server.py`）
工作设备（Worker Device）	接入中枢、在本地进程池中隔离运行步骤并通过 QUIC 收发中间产物的工作进程（`run_device.py`）
算法 / 步骤（Algorithm / Step）	遵循统一类签名的 Python 执行单元，可读取上游产物并上报业务输出与实测质量
DAG（有向无环任务图）	描述多个步骤之间先后依赖关系的拓扑结构；互不依赖的分支可跨设备并行执行
应用与作业（Application & Job）	应用是绑定了清单、步骤与旋钮的可复用流程模板；作业（`job_id`）是某次具体调用产生的执行实例

1.3 资源与应用的双向自适应

OpenSAI 将两个互补方向的自适应放入同一个运行期闭环中协同求解：

1. **资源适配应用（方向一）**：依据步骤的计算特征、数据位置、可并行比例与实时负载，为每一步挑选合适的执行设备、CPU 核数、GPU 目标及传输路径。
2. **应用适配资源（方向二）**：依据调用方给定的截止时间（`deadline`）或质量要求，在应用允许的“旋钮（Knobs）”候选档位中动态选择分辨率、采样比例、特征维数或协作通信量。

每一次步骤执行完成后，实测耗时都会自动回流更新算法在该类硬件上的**性能画像（Performance Profile）**。你可以用 `-k` 锁定业务必须固定的参数，将其余弹性参数交给平台自动权衡。

1.4 四种等价交互入口

交互入口	典型适用场景
双语 Web 控制台	可视化浏览三维集群拓扑、交互编排 DAG、对比预测与实测表现、审查决策解释
命令行 `opensai`（CLI）	终端日常提交、自动化 Shell / PowerShell 批处理脚本、远程 SSH 运维

交互入口	典型适用场景
Python SDK（`OpenSAIClient`）	将分布式计算无缝嵌入现有后端服务、数据流水线或自动化评测程序
MCP 服务与使用 Skill	授权 Claude、Cursor、OpenClaw 等智能体理解自然语言意图并代为操控集群

四个入口连接同一个协调中枢与应用注册表——通过 CLI、SDK 或智能体提交的作业，均可在 Web 控制台中实时查看。

2. 安装与本地运行

2.1 运行环境与依赖分层

- **基础要求**：Python 3.10 及以上版本（Windows 10/11、Ubuntu / Debian 等主流 Linux 及 macOS）。发布包已内置编译好的前端静态资源，正常使用**无需安装 Node.js、Docker 或 Kubernetes**。
- **按需扩展依赖**：核心环境仅包含 Redis 通信、Flask 桥接、系统资源采集与 QUIC 协议库；`numpy`、OpenCV、PyTorch 及 MCP SDK 仅在实际运行对应示例或接入智能体时按需安装。

2.2 一次性环境初始化

Linux / macOS：

```bash
bash scripts/setup.sh
source .venv/bin/activate
```

Windows PowerShell：

```powershell
powershell -ExecutionPolicy Bypass -File scripts\setup.ps1
.venv\Scripts\Activate.ps1
```

安装脚本会自动创建 `.venv`、安装核心运行依赖与 `opensai` 客户端，并复用系统已有 Redis 或自动准备便携版 Redis（Linux / macOS 若需从源码编译便携 Redis，需具备 `cc`、`make` 及 `curl` / `wget`；Ubuntu/Debian 需预装 `python3-venv`）。可通过 `OPENSAI_PYTHON` 指定解释器路径，通过 `PIP_INDEX_URL` 指定国内 PyPI 镜像源。

如需安装可选依赖，支持直接使用脚本开关或单独调用 `pip`：

```bash
# 方式一：使用安装脚本内置扩展参数
bash scripts/setup.sh --agent    # 安装 MCP 智能体服务依赖（PowerShell 用 -Agent）
bash scripts/setup.sh --ml       # 安装 PyTorch / TorchVision 深度学习依赖（PowerShell 用 -Ml）

# 方式二：在已激活的 .venv 中按需安装
python -m pip install numpy
python -m pip install -r Offloading-System/requirements-optional.txt
python -m pip install -r Offloading-System/requirements-ml.txt
python -m pip install -r Offloading-System/requirements-agent.txt
```

2.3 启动、状态检查与停止

Linux / macOS：

```bash
bash scripts/run.sh --with-web
opensai devices list
opensai cluster status
```

Windows PowerShell：

```powershell
powershell -ExecutionPolicy Bypass -File scripts\run.ps1 -WithWeb
opensai devices list
opensai cluster status
```

访问 `http://127.0.0.1:5000` 打开控制台。注意：`run` 脚本具有幂等重启特性，重复执行会先平滑停止旧的本机中枢与设备进程再重新拉起，请在作业间隙执行。

停止本机服务（附加 `--all` / `-All` 一并关闭由 OpenSAI 脚本托管的便携 Redis）：

- **Linux / macOS：** `bash scripts/stop.sh --all`
- **Windows PowerShell：** `powershell -ExecutionPolicy Bypass -File scripts\stop.ps1 -All`

2.4 单独启动 Web 控制台桥接

当中枢与设备已在后台运行、仅需单独拉起本地 `:5000` Web 桥接界面时，可执行 `bash scripts/start_console.sh`（Windows 执行 `powershell -ExecutionPolicy Bypass -File scripts\start_console.ps1` 或双击 `start-console.bat`；支持 `--build` 强制重新构建前端与 `--no-browser` 不自动弹窗）。默认仅监听本机 `127.0.0.1:5000`；如需在可信局域网其他机器访问，可设置 `OPENSAI_BRIDGE_HOST=0.0.0.0` 或使用 SSH 转发（`ssh -L 5000:127.0.0.1:5000 <中枢主机>`）。即便尚未连接任何真实集群，也可在地址后附加 `?offline`（或点击提示横幅 **“用演示数据体验”**）完整体验全部界面。

3. 部署多机集群

3.1 规划节点角色与启动命令

挑选一台各工作节点均可访问的机器担任**协调中枢（Hub）**（默认 `run` 脚本也会在中枢机器上顺带启动 1 个本地工作设备进程）；其余边缘机器仅需启动工作设备进程 `run_device.py`。

假设中枢机器的局域网 IP 为 `192.168.1.10`：

第 1 步 — 在中枢机器启动：

```bash
export OPENSAI_HOST=192.168.1.10
bash scripts/run.sh --with-web
```

第 2 步 — 在每台远端工作设备完成 `setup` 后启动：

Linux / macOS：

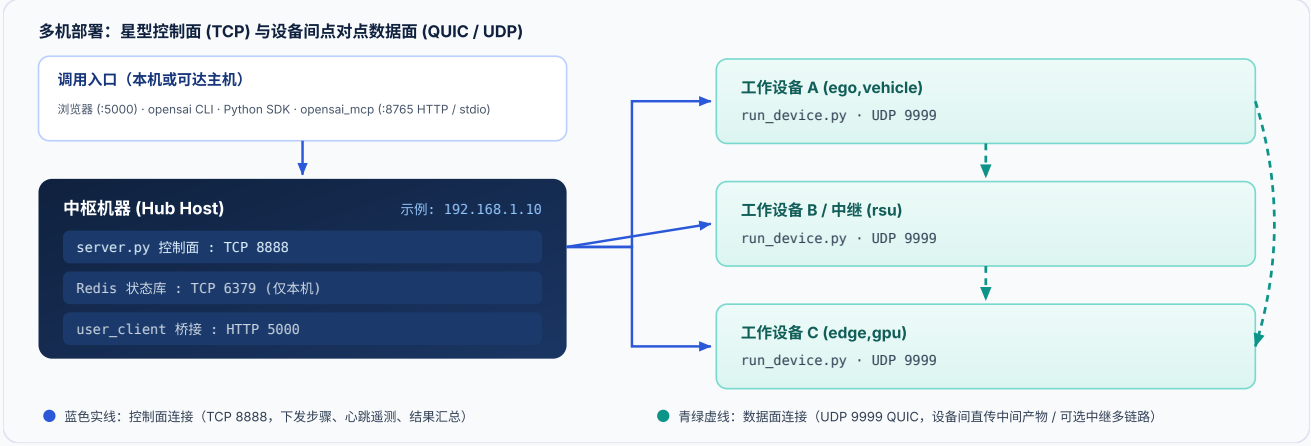
```bash
source .venv/bin/activate
export OPENSAI_HOST=192.168.1.10
python Offloading-System/dag_offload/run_device.py
```

Windows PowerShell：

```powershell
$env:OPENSAI_HOST="192.168.1.10"
.venv\Scripts\python.exe Offloading-System\dag_offload\run_device.py
```

POWERSHELL

远端工作设备无需运行 Redis（Redis 仅驻留在中枢本机）。



多机星型控制面与点对点 QUIC 数据面拓扑

3.2 端口矩阵与网络安全边界

服务组件	默认协议与端口	需要访问该端口的对象
中枢控制面（`server.py`）	`TCP 8888`	所有工作设备、CLI、Python SDK、MCP 服务及 Web 桥接进程
设备数据面（QUIC）	`UDP 9999`	需要跨设备拉取中间产物或承担中继转发的其他工作设备
Redis 状态库	`TCP 6379`	仅限中枢本机内部进程访问
Web 控制台桥接	`HTTP 5000`（默认 `127.0.0.1`）	本机浏览器（跨机访问需在可信网络显式设置 `OPENSAI_BRIDGE_HOST` 或走 SSH 转发）
远程 MCP HTTP 服务	`HTTP 8765`（路径 `/mcp`）	远端 MCP 智能体客户端（默认强制校验 Bearer 令牌）

> ▲ **安全与边界提醒**
>
> **网络连通与安全边界（必读）：**
>
> 1. **控制面安全：** `TCP 8888` 控制端口面向受控局域网设计，本身不内置公网身份鉴权，**切勿直接映射到公网**；跨异地局域网组网时，请先使用 Tailscale、WireGuard 或企业 VPN 建立加密私网。
> 2. **数据面双向互通：** 工作设备能连上中枢 `TCP 8888`，不代表工作设备彼此之间的 `UDP 9999` 已放通。多机流水线传递中间产物时，务必放行设备间的 `UDP 9999`（或自定义 `OPENSAI_DATA_PORT`）防火墙规则。

3.3 配置设备本地数据目录与角色标签

启动工作设备前，可通过环境变量指定数据集根目录与角色标签：

```bash
export OPENSAI_DATA_DIR=/data/opensai
export OPENSAI_DEVICE_TAGS=edge,gpu
python Offloading-System/dag_offload/run_device.py
```

- `OPENSAI_DATA_DIR` 下的**直接子文件或直接子目录**会被自动扫描并作为本地数据集名称上报给中枢（若数据在其他磁盘，创建软链接放入该目录即可）。
- `OPENSAI_DEVICE_TAGS` 为设备附加角色语义标签（如 `ego`、`vehicle`、`rsu`、`edge`、`gpu`），供应用清单中的 `run_on` 规则优先匹配。

3.4 单机模拟多工作设备进程

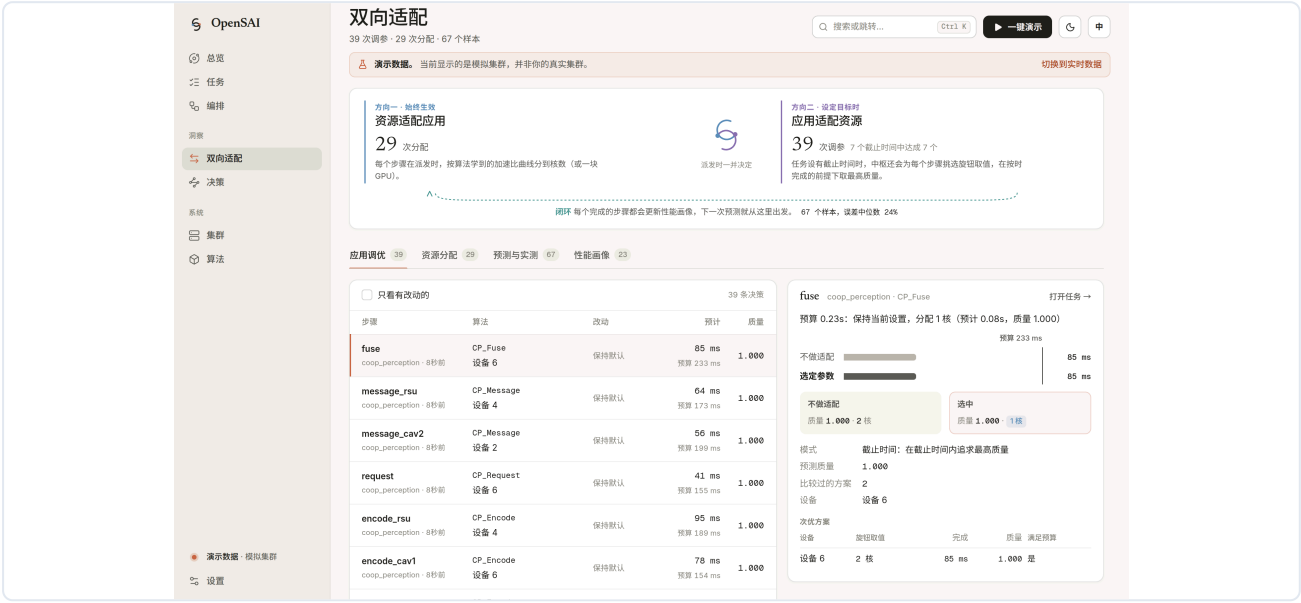
在单台电脑上测试多设备路由或中继传输时，可为不同工作进程分配独立的 `OPENSAI_DATA_PORT` 与 `OPENSAI_ARTIFACTS_DIR`（并可按需设置 `OPENSAI_DEVICE_CPUS` 划分逻辑核数）以隔离端口与落盘目录。注意单机多进程共享同一物理主机硬件，正式评估吞吐与时延时请使用真实多机环境。

4. 使用双语 Web 控制台

4.1 语言、纸墨主题与实时 / 演示数据模式

浏览器打开 `http://127.0.0.1:5000`：

- **双语与纸墨主题切换**：控制台默认采用英文界面与暖象牙色 **“纸”(Paper)** 浅色主题；点击顶栏右侧 `EN` / `中` 按钮可随时切换中英文，点击旁侧月亮/太阳图标可切换 **“纸”(浅色)** 与 **“墨”(Ink 深色)** 主题，偏好自动保存在当前浏览器中。顶栏还提供全局命令面板（`Ctrl K` / `⌘K`）与 **“一键演示”(Run demo)** 按钮。
- **区分“实时”与“演示数据”**：查看左下角连接状态指示器，正常连接真实中枢时显示 **“已连接 · 实时”(`Connected · live`)**；若通过 `?offline` 或在 **“设置”** 中开启了模拟集群预览，顶部会显示醒目的 **“演示数据”** 提示横幅——点击横幅右侧 **“切换到实时数据”** 即可回到真实集群视图。



OpenSAI 控制台双向适配与性能画像页面

4.2 八大功能页面导览

左侧导航名称（对应英文界面）	核心用途
总览（`Overview`）	二维轨道拓扑图与沉浸式 **`3D 视图`**（**`3D view`**）展示在线设备、可用核心、在飞任务、截止时间达成率与实时决策流
任务（`Runs`）	汇总来自控制台、CLI、SDK 与智能体的全部作业，查看执行甘特时间线、逐节点输出与耗时
编排（`Compose`）	可视化拖拽编排算法节点、导入/导出 JSON、设定目标提交，并通过 **应用**（**`Apps`**）标签直接运行已注册应用
双向适配（`Adaptation`）	审查应用参数调优决策、分核与 GPU 资源分配、预测与实测对比散点图及实测拟合的**性能画像**（**`Profiles`**）
决策（`Decisions`）	集中回溯每一步的设备放置打分、四维资源预留、QUIC 直连/中继传输路径与掉线恢复事件
集群（`Cluster`）	按卡片或表格查看工作设备算力利用率，以及逐设备承诺与归还的四维**资源账本**（**`Resource ledger`**）
算法（`Algorithms`）	浏览内置与新增算法的输入参数、可调旋钮、声明先验画像与实测加速比，支持上传自定义算法文件
设置（`Settings`）	切换纸/墨主题与界面语言、配置桥接地址、开关模拟演示数据，以及查阅平台启动**系统参数**（**`System parameters`**）

4.3 推荐日常操作与排障动线

日常使用时，先在 **总览**（或 **集群**）确认设备在线 → 通过 CLI/SDK 或 **编排**（含已注册应用页签）提交任务 → 在 **任务** 查看输出与端到端耗时 → 若想深究某次配置选择，打开 **双向适配** 或 **决策** 对照同一作业记录。若作业失败，在 **任务** 页点开失败步骤即可直接看到缺库、缺数据或异常报错详情。

5. 运行和管理应用

5.1 查询与注册应用

```bash
opensai app list
opensai app info cluster_analytics
opensai app validate examples/apps/cluster_analytics
opensai app register examples/apps/cluster_analytics
opensai app run cluster_analytics -i points=30000 --wait
```

`app info` 会清晰列出该应用接受的 `inputs`、可调 `knobs`、步骤拓扑及默认 `qos`。`validate` 在本地完成语法与清单校验；`register` 将清单与 `steps/` 源码提交至中枢并自动分发给工作设备（跨机远程执行管理操作时使用中枢配置的 `OPENSAI_ADMIN_TOKEN`）。注意：第三方 Python 库、外部模型权重与大数据集需预先在对应工作设备上准备就绪。

5.2 传入业务输入、锁定旋钮与指定目标

CLI 选项	调用示例	作用说明
业务输入 `-i`	`-i points=30000 -i seed=1`	指定本次运行的业务数据规模或场景参数

CLI 选项	调用示例	作用说明
锁定旋钮 `-k`	`-k dims=16`	显式固定某个可调旋钮，本次运行不再由平台改动它
截止时间 `--deadline`	`--deadline 3`	设定端到端目标完成时间（单位：秒）
优化模式 `--mode`	`--mode fastest`	选择 `deadline`、`fastest` 或 `efficient` 目标策略
质量下限 `--min-quality`	`--min-quality 0.9`	在该应用声明的质量尺度上要求最低可接受质量
同步等待 `--wait`	`--wait --timeout 120`	在终端阻塞等待作业完成并返回完整结果

调用时传入的选项会覆盖应用清单中的同名默认值；若应用清单本身已声明默认 `deadline_s`，省略 `--deadline` 时仍会沿用清单默认时限。

5.3 版本管理与注销

更新应用代码与 `app.yaml` 中的 `version` 后重新 `register` 即可平滑升级同名应用（附加 `--no-replace` 可防止覆盖已有同名应用）：

```bash
opensai app register my_app --no-replace
opensai app run my_app --app-version 1.0 --wait
opensai app unregister my_app
```

注销（`unregister`）会移除中枢注册记录与同步代码，请在无在途作业时执行，并妥善保留本地应用包源码以便随时重新注册。

5.4 五个内置示例应用选型指南

应用名称	演示重点	环境与数据准备
`number_stats`	最小标准库应用、跨步产物传递、采样率旋钮	零额外第三方依赖（完整源码见第 6.2 节）
`cluster_analytics`	双分片并行 K-Means 聚类、参数调优与中心合并	安装 `numpy`；自动生成本地合成点集
`video_counting`	合成视频流双分支目标检测计数、分辨率与抽帧联合选择	安装 `numpy`；内置合成帧生成器
`coop_perception`	自车/协作车/路侧三源点云本地性、选择性通信 `comm` 与融合	安装 `numpy`；附带本地场景生成与逐帧回放脚本
`how2comm_perception`	基于 How2comm 模型的真实路口激光雷达边缘 GPU 协同检测	边缘 GPU 节点安装 PyTorch 相关依赖、How2comm 源码权重及 V2XSet 场景

5.5 实战演练：运行多源协同感知（`coop_perception`）

该应用模拟自车（`cav1`）、协作车（`cav2`）与路侧单元（`rsu`）协同感知周边目标：各方在本地读取并筛选点云，再将精选目标信息发往融合步骤计算综合检测结果（`tp`、`fp`、`fn` 与 `f1`）。

1. 单机生成场景并运行单帧：

```bash
python -m pip install numpy
python examples/apps/coop_perception/make_scene.py Offloading-System/data --name syn --frames 20
opensai app register examples/apps/coop_perception
opensai data list
opensai app run coop_perception -i scene=syn -i frame=0 --wait
```

（若自定义了 `OPENSAI_DATA_DIR`，请将生成路径改为该目录；确认 `opensai data list` 出现 `syn_cav1`、`syn_cav2`、`syn_rsu` 后即可执行。）

2. 对比固定通信比例与截止时间自动调节：

旋钮 `comm` 控制协作方发送目标信息的比例（候选值 `0.1, 0.25, 0.5, 1.0`）：

```bash
# 固定不同上传比例做对照
opensai app run coop_perception -i scene=syn -i frame=0 -k comm=1.0 --wait
opensai app run coop_perception -i scene=syn -i frame=0 -k comm=0.25 --wait

# 去掉 -k，交由平台按 0.8 秒目标自动权衡
opensai app run coop_perception -i scene=syn -i frame=0 --deadline 0.8 --wait
```

3. 连续帧率回放与多机分布：

```bash
python examples/apps/coop_perception/replay.py --scene syn --fps 2 --frames 0-19 --json coop-run.json
```

扩展到多机时，只需将 `syn_cav1`、`syn_cav2`、`syn_rsu` 分别放入自车（标签 `ego,vehicle`）、协作车（`vehicle`）与路侧设备（`rsu`）的数据目录，读取步骤即会自动跟随数据就近执行。

5.6 进阶：复现真实 GPU 模型版协同检测（`how2comm_perception`）

若需在带 GPU 的边缘服务器上体验真实三维激光雷达模型推理（详见 `examples/apps/how2comm_perception/README.md`）：

1. 在项目目录克隆 `third_party/How2comm` 源码，并将 V2XSet 预训练 `config.yaml` 与 `net_epoch32.pth` 放入 `pre-train/v2xset/`（步骤内置纯 PyTorch / NumPy 等价算子，免编译 CUDA 扩展）；
2. 在边缘 GPU 节点安装 `requirements-ml.txt` 及 `einops shapely icecream matplotlib`，设置角色标签 `OPENSAI_DEVICE_TAGS=edge`；
3. 使用 `prepare_datasets.py` 将 V2XSet 场景切分为 `v2xraw_cav1`、`v2xraw_cav2`、`v2xraw_rsu` 分置各端，注册应用后即可通过 `opensai app run how2comm_perception -i frame=40 --wait` 运行（协作车离线时可选分支自动跳过）。

6. 编写可复用应用

6.1 应用包目录规范与容量限制

一个标准应用包由根目录下的 `app.yaml`（或 `app.json`）清单，以及 `steps/` 子目录下的 `Algorithm_<名称>.py` 步骤文件构成：

- 文件名、类名与模块导出的 `__all__[0]` 必须保持一致（推荐加应用专属前缀如 `Algorithm_MyApp_Load`，防止跨应用命名冲突）；
- 每个步骤文件应保持自包含，可导入 Python 标准库及执行设备已安装的第三方包；
- 单个应用包最多包含 **32 个**步骤文件，单文件上限 **1 MB**，步骤源码总计不超过 **3 MB**。

6.2 完整可运行范例：数值采样统计（`number_stats`）

该示例已随附在 `examples/apps/number_stats`，可先直接校验运行，再对照阅读源码：

```bash
opensai app validate examples/apps/number_stats
opensai app register examples/apps/number_stats
opensai app run number_stats --wait
```

1. 应用清单 `number_stats/app.yaml`：

```yaml
name: number_stats
version: "1.0"
title: Number statistics
description: Generate numbers and calculate a sampled mean.
inputs:
  count:
    type: integer
    default: 10000
    minimum: 100
    maximum: 100000
knobs:
  sample:
    description: Fraction of input values included in the summary.
    values: [0.25, 0.5, 1.0]
    default: 1.0
    quality: [0.25, 0.5, 1.0]
    bind: [summarize.sample]
steps:
  - name: generate
    algorithm: Algorithm_NumberStats_Generate
    params: {count: "${count}"}
  - name: summarize
    algorithm: Algorithm_NumberStats_Summarize
    after: [generate]
output: summarize
qos:
  deadline_s: 3
```

2. 上游生成步骤 `number_stats/steps/Algorithm_NumberStats_Generate.py`：

```python
import json

__all__ = ["Algorithm_NumberStats_Generate"]

class Algorithm_NumberStats_Generate:
    def __init__(self, param_dict, data_dict):
        self.param = param_dict or {}

    def run(self):
        count = int(self.param.get("count", 10000))
        values = [i % 101 for i in range(count)]
        with open(self.param["_artifact_out"], "w", encoding="utf-8") as stream:
            json.dump(values, stream)
        return f"Generated {count} values"
```

3. 下游汇总步骤 `number_stats/steps/Algorithm_NumberStats_Summarize.py`：

```python
import json

__all__ = ["Algorithm_NumberStats_Summarize"]

class Algorithm_NumberStats_Summarize:
    def __init__(self, param_dict, data_dict):
        self.param = param_dict or {}

    def run(self):
        values = []
        for path in (self.param.get("_artifact_in") or {}).values():
            with open(path, encoding="utf-8") as stream:
                values.extend(json.load(stream))
        if not values:
            return "ERROR: No input values"
        sample = float(self.param.get("sample", 1.0))
        count = max(1, int(len(values) * sample))
        picked = values[:count]
        mean = sum(picked) / len(picked)
        self.quality = len(picked) / len(values)
        self.metrics = {"samples": len(picked), "mean": mean}
        with open(self.param["_artifact_out"], "w", encoding="utf-8") as stream:
            json.dump(self.metrics, stream)
        return f"mean={mean:.4f}, samples={len(picked)}"
```

6.3 清单核心字段速查

顶层字段	编写规范与用途
`name`	小写字母开头，仅含小写字母、数字与下划线，长度 3～40 字符
`version`	应用版本字符串（如 `"1.0"`）
`title`、`description`	供控制台用户与 MCP 智能体理解应用功能的简明标题与说明
`requires`	步骤运行所需的第三方 Python 包名列表（用于提示与检查）
`inputs`	业务输入定义，支持 `integer`、`number`、`string`、`boolean`、`array`、`object` 及 `default`、`minimum`、`maximum`、`enum`
`knobs`	允许平台动态调节的参数候选值、默认值、绑定关系及质量/开销表
`steps`	步骤列表（算法类名、参数模板 `${input}`、`after` 前驱依赖、数据与标签约束）
`output`	作为整个应用主输出结果的步骤名称（必须为非可选步骤）
`qos`	应用级默认运行目标（调用时未覆盖的字段沿用此处配置）

6.4 声明可调旋钮（Knobs）

在 `knobs.<名称>` 下，`values` 列出离散候选档位，`default` 指定默认档，`bind` 将旋钮绑定到一个或多个 `步骤名.参数名`。可选的 `quality`、`cost`、`out` 数组分别按与 `values` 相同的顺序给出各档位的相对业务质量（`0`～`1`）、相对计算开销与相对输出体积先验（实际运行耗时将由性能画像持续在线学习校准）。当多条并行分支必须同步采用同一档位（例如左右两路提取相同的特征维数）时，设置 `tie: true`。

6.5 运行时注入参数与返回约定

平台自动注入键	在 `run(self)` 中的使用方式
`_artifact_in`	`{前驱节点ID: 本地可读文件路径}` 映射字典；多前驱时请在产物内容中记录来源标识，勿依赖遍历顺序
`_artifact_out`	当前步骤写给下游的本地目标文件路径（使用 `numpy.save` 等库时请传入已打开的文件对象，避免自动追加 `.npy` 扩展名导致找不到产物）
`_data_dir`	当前工作设备的本地数据集根目录路径
`_alloc`	当前步骤分得的资源字典（如 `threads`、`core_ids`、GPU 设备信息与内存上限）

`run()` 返回的字符串即为节点业务输出（以 `ERROR` 开头将被判定为步骤执行失败）；通过 `self.quality` 上报 `0` ～ `1` 实测质量，通过 `self.metrics` 上报结构化数值字典。

6.6 数据本地性、角色标签与可选容错分支

- **强制跟随数据（`data: [数据集名]`）**：限定该步骤只能在拥有指定本地数据集的设备上运行。
- **偏好设备角色（`run_on: [ego, edge]`）**：当集群存在任一匹配标签的在线设备时优先收窄候选池；若当前无匹配标签设备在线，则自动放宽以保障可运行性（因此硬性数据绑定请始终使用 `data`）。
- **可选协作分支（`optional: true`）**：当该步骤因设备离线、缺数据、异常或主动返回 `SKIP:` 前缀而跳过时，下游必需融合步骤仍可使用其余已完成分支的输入继续运行。

6.7 配合资源分配与预热接口

多线程/科学计算步骤应读取 `self.param["_alloc"]` 中的 `threads` 或 `core_ids` 对齐内部并行线程数；模型推理步骤可定义静态方法 `warmup()` 预加载权重以消除首帧冷启动开销，也可在类中提供可选 `PROFILE` 字典补充并行度与 GPU 能力先验。

7. 直接编排与提交 DAG

7.1 临时组合已有算法

对于一次性实验或临时拼装内置算法的场景，无需封装完整应用包，直接编写 `dag_spec` JSON 即可。以下示例并行运行两个采样规模不同的圆周率计算，完成后汇合：

```json
{
  "name": "parallel-pi",
  "nodes": [
    {
      "id": 1,
      "name": "pi-a",
      "algorithm": "Algorithm_Pi_Calculation",
      "params": {"count_num": 200000}
    },
    {
      "id": 2,
      "name": "pi-b",
      "algorithm": "Algorithm_Pi_Calculation",
      "params": {"count_num": 300000}
    },
    {
      "id": 3,
      "name": "join",
      "algorithm": "Algorithm_Sleep_1",
      "params": {"sleep_time": 0.1},
      "before_nodes": [1, 2]
    }
  ]
}
```

保存为 `parallel-pi.json` 后执行 `opensai dag validate -f parallel-pi.json` 与 `opensai dag submit -f parallel-pi.json`。注意：节点 `id` 须为唯一正整数，`before_nodes` 引用前驱节点 `id` 且整张图不得包含环路。

7.2 节点级资源声明与设备绑定

节点对象支持通过 `resources` 请求四维配额：`cores`（整数核数）、`mem_mb`（内存 MiB）、`storage_mb`（存储 MiB）以及 `bw_mbps`。

> ◆ **重要约定**
>
> **带宽字段单位说明**：出于兼容历史接口约定，节点 `resources` 中的 `bw_mbps` 字段在当前版本实际按 **`MB/s`（兆字节每秒）** 解释与限速，填写时请勿按 `Mbps`（兆比特每秒）换算以免放大 8 倍。

如需将某节点固定到特定机器，将 `opensai devices list` 返回的 `allocation_index` 填入 `direct_allocation_id`（默认 `-1` 表示交由平台智能放置）。节点 `params` 中还可使用 `requires_data` 声明所需本地数据集，使用 `adapt: false` 或 `knob_lock` 锁定调参（以下划线 `_` 开头的字段由平台保留）。

7.3 可拆分算法的自动结构分片

声明了 `PROFILE.split` 的可分片算法，允许调用方在节点 `params` 中传入 `split: "auto"`，由中枢根据当前在线设备数量自动展开并行分片子任务与归并节点。

8. 选择时间、质量与资源目标

8.1 三种 QoS 优化模式

模式（`mode`）	调度优化目标	必填与可选字段
`deadline`	在给定截止时间内尽可能保留更高的应用业务质量	必填 `deadline_s`

模式（`mode`）	调度优化目标	必填与可选字段
`fastest`	在满足质量底线的前提下追求最短完成时间	可选 `min_quality`
`efficient`	在满足时间与质量要求的前提下尽量降低核秒与资源消耗	必填 `deadline_s`，可选 `min_quality`

此外，QoS 支持整数 `priority`（数值越大优先级越高）。注意 `min_quality`（如 `0.9`）对应应用自身声明的相对质量尺度，不等同于任意视觉任务的绝对 90% 识别准确率。

8.2 放置模式切换：`plan` 与 `adaptive`

放置模式	决策求解顺序	适用场景
`plan`（默认）	先基于数据本地性与链路做整图设备放置规划，再按 QoS 为各节点选核数与旋钮	常规部署，便于清晰观察整图数据亲和与放置关系
`adaptive`	将候选设备、CPU/GPU 目标、分配核数与可调旋钮放入同一搜索空间联合比较	希望设备选择本身也随截止时间紧迫度动态权衡

在中枢启动前设置 `OPENSAI_PLACEMENT` 并重启本机或中枢服务即可切换（请先等待在途任务完成）：

Linux / macOS：

```bash
export OPENSAI_PLACEMENT=adaptive
bash scripts/run.sh --with-web
```

Windows PowerShell：

```powershell
$env:OPENSAI_PLACEMENT="adaptive"
powershell -ExecutionPolicy Bypass -File scripts\run.ps1 -WithWeb
```

恢复默认时改回 `plan` 并重启中枢。按需分核（`OPENSAI_AUTO_CORES=1`）默认开启；基于历史画像的内存与存储自动定档可分别通过 `OPENSAI_AUTO_MEM=1` 和 `OPENSAI_AUTO_STORAGE=1` 开启。所有配置均在步骤派发前锁定，后续未派发步骤可根据最新负载动态重选。

9. 数据管理与多链路传输

9.1 准备与上传数据集

工作设备默认扫描 `Offloading-System/data`（或 `OPENSAI_DATA_DIR`），将其下的直接子目录或文件上报至 `opensai data list`。小体积测试文件可直接通过命令行分发：

```bash
opensai data upload ./sample_data --name sample_data --device 1
```

`--device` 接收设备的 `allocation_index`（可多次指定；省略则广播上传至全部在线设备；覆盖同名数据需加 `--replace`）。大体积数据集推荐直接拷入或软链接至目标设备的 `OPENSAI_DATA_DIR`。

9.2 QUIC 直传与开启 DELTA 中继多链路

同机上下游步骤直接读取本地落盘产物；跨设备步骤默认通过点对点 QUIC（`UDP 9999`）拉取产物。当直连链路受限且集群中存在至少第 3 台空闲可达设备时，可在中枢启动前开启统一中继多链路规划：

```bash
export OPENSAI_MULTIPATH=1
```

重启中枢生效（设为 `0` 关闭）。`DELTA` 规划器会综合传输任务时限、路径建连开销、实测带宽与共享瓶颈，利用多条标准 QUIC 连接并行组合直连路径与中继路径（无需内核支持 MPTCP 或 MPQUIC），并在传输过程中根据实时进度动态调整份额。

9.3 理解传输记录与产物清理

启用总开关后，若单次产物较小、辅助路径收益不足或所有路径共用同一饱和网卡出口，规划器仍会合理选择单路直连。作业级 `bytes_moved` 统计接收端拉取的**有效载荷字节数**；分析多跳中继带来的额外网络流量时，请在控制台 **“决策”(Decisions)** 页查看传输路径明细。中间产物可通过设备端的 `OPENSAI_ARTIFACT_TTL_S` 与 `OPENSAI_ARTIFACT_MAX_MB` 控制保留时长与容量水位。

10. 查看结果和性能

10.1 查询作业与决策解释

```bash
opensai result get YOUR_JOB_ID
opensai result wait YOUR_JOB_ID --timeout 120
opensai explain YOUR_JOB_ID
```

结果对象顶层包含：`status`（作业状态）、`duration_s`（端到端秒数）、`nodes`（各节点设备、核数、输出与状态）、`qos`（目标与实际达成情况）、`efficiency`（资源效率统计）、`app`（应用版本与选中旋钮摘要）以及失败时的 `error`。

10.2 关键统计指标口径

- `duration_s`：从提交到终态的端到端墙钟时间。
- `compute_s`、`fetch_s`、`queue_s`：所有节点的计算、跨机拉取与排队时间累计值；含并行分支时各分支时间重叠，**不可直接相加充当用户等待时长**。
- `core_seconds` 与 `cpu_alloc_eff`：前者为 CPU 节点“分配核数 × 计算耗时”累计核秒，后者为“实际 CPU 时间 ÷ 分配核秒”，用于衡量多核分配是否被算法真正利用（GPU 节点不计入该 CPU 比值，`gpu_tasks` 单独统计 GPU 执行节点数）。

进行严谨性能评估时，请固定输入规模、应用版本、设备集合与网络条件重复多次运行，使用 SDK 将 `result.to_dict()` 保存为归档 JSON，同时考察完成时间、业务质量与资源消耗。

11. Python SDK

11.1 客户端连接与常用方法

```python
from opensai_client import OpenSAIClient
from opensai_contracts import OpenSAIConfig

config = OpenSAIConfig(server_host="127.0.0.1", server_port=8888, timeout=30.0)
with OpenSAIClient(config) as client:
    print(client.list_devices())
```

`OpenSAIClient` 采用惰性连接机制，配合 `with` 上下文可在退出时自动释放连接。核心方法速查：

SDK 方法	用途说明
`list_devices()`、`cluster_status()`	获取在线工作设备列表与集群总体状态
`list_algorithms()`、`get_catalog()`	查询底层算法目录、参数契约与旋钮定义
`validate_dag(dag)`、`submit_dag(dag, qos=...)`	本地校验并提交自定义 DAG
`submit_task(algorithm, params=..., wait=True)`	快速调用单个算法
`list_apps()`、`get_app(name)`	发现已注册应用及其清单详情
`validate_app(path)`、`register_app(path)`、`unregister_app(name)`	校验、注册或移除应用包
`run_app(name, inputs=..., knobs=..., qos=...)`	按输入、固定旋钮与 QoS 目标运行应用
`get_result(job_id)`、`wait_result(job_id, timeout=...)`	查询作业快照或等待终态
`explain_job(job_id)`	获取设备放置、资源与旋钮决策解释
`list_data()`、`upload_data(path, devices=...)`	查询设备数据集或上传小文件
`get_metrics(last_n=12)`、`get_events()`	拉取集群近期统计样本与运行时事件流

11.2 完整调用、持久化保存与异常处理

```python
import json
from opensai_client import OpenSAIClient

with OpenSAIClient(server_host="127.0.0.1") as client:
    job = client.run_app(
        "cluster_analytics",
        inputs={"points": 30000, "seed": 1},
        qos={"deadline_s": 3, "mode": "deadline"},
    )
    result = client.wait_result(job.job_id, timeout=120)
    with open("result.json", "w", encoding="utf-8") as stream:
        json.dump(result.to_dict(), stream, ensure_ascii=False, indent=2)
    print(result.status, result.duration_s)
```

在需要自动重试的网络环境中，可向提交方法传入唯一 `idempotency_key`（同一次业务请求复用同一键，新请求生成新键）。注意区分 `wait_result` 客户端等待超时（作业仍在后台运行，继续轮询原 `job_id` 即可）与 `RemoteError` / 节点 `FAILED` 业务异常。

12. CLI 速查

全局连接参数须置于子命令之前，例如：`opensai --host 192.168.1.10 --port 8888 devices list`（常用全局选项还包括 `--timeout`、`--poll-interval` 与 `--config`）。

操作目的	标准 CLI 命令
查看在线设备 / 集群健康	`opensai devices list` / `opensai cluster status`
浏览底层算法目录 / 近期统计	`opensai catalog` / `opensai metrics --last-n 12`
校验并提交 JSON DAG	`opensai dag validate -f job.json` / `opensai dag submit -f job.json`
等待结果 / 查看决策解释	`opensai result wait YOUR_JOB_ID --timeout 120` / `opensai explain YOUR_JOB_ID`
初始化 / 查看应用包	`opensai app init my_app` / `opensai app list` / `opensai app info my_app`
注册 / 运行应用	`opensai app register my_app` / `opensai app run my_app -i size=10000 --wait`
查看 / 上传设备数据集	`opensai data list` / `opensai data upload sample.csv --device 1`

在 PowerShell 或包含复杂嵌套 JSON 参数时，可使用 `@文件路径` 直接读取参数文件（如 `params.json` 内容为 `{"count_num": 200000}`）：

```bash
opensai task submit Algorithm_Pi_Calculation -p @params.json
```

13. 智能体接入

OpenSAI 原生提供 `opensai_mcp` 服务（支持本地 `stdio` 与带 Bearer 鉴权的远程 `Streamable HTTP`）及中英文等价的 `opensai/SKILL.md` 技能文件。执行 `bash scripts/setup.sh --agent` 并通过 `python -m opensai_mcp --check` 后即可接入主流 AI 客户端，完整配置步骤、工具清单与团队限流策略见 智能体接入指南 与 使用 Skill 规范。

14. 常用运行配置速查

所有进程级环境变量均在启动对应进程（中枢或工作设备）前设置，修改后重启对应进程生效（Bash 使用 `export 变量=值`，PowerShell 使用 `$env:变量="值"`）：

环境变量	默认值	生效进程与作用说明
`OPENSAI_HOST`	`127.0.0.1`	中枢监听地址，以及设备/客户端连接的目标中枢地址
`OPENSAI_PORT`	`8888`	中枢控制面 TCP 端口
`OPENSAI_BRIDGE_HOST`	`127.0.0.1`	Web 控制台桥接（`:5000`）监听地址；可信局域网跨机访问可设为 `0.0.0.0`
`OPENSAI_DATA_PORT`	`9999`	工作设备 QUIC 数据面 UDP 端口
`OPENSAI_DATA_DIR`	`Offloading-System/data`	工作设备扫描本地数据集的根目录
`OPENSAI_DEVICE_TAGS`	空	逗号分隔的工作设备角色标签（如 `ego,vehicle`、`rsu`、`edge,gpu`）

环境变量	默认值	生效进程与作用说明
`OPENSAI_PLACEMENT`	`plan`	中枢放置模式；设为 `adaptive` 启用设备+资源+旋钮联合选择
`OPENSAI_AUTO_CORES`	`1`	按算法加速比画像与空闲核自动分配 CPU 核数
`OPENSAI_MULTIPATH`	`0`	设为 `1` 启用 `DELTA` 统一 QUIC 中继多链路规划
`OPENSAI_TASK_WORKERS`	`4`	每台工作设备的并发执行进程池槽位数
`OPENSAI_GPU_SLOTS_PER_GPU`	`1`	单块 GPU 允许同时并发执行的任务槽位数
`OPENSAI_JOB_TTL_S`	`3600`	已完成作业元数据在内存账本中的保留秒数
`OPENSAI_ARTIFACT_TTL_S`	`3600`	工作设备本地中间产物的定时回收保留秒数
`OPENSAI_ADMIN_TOKEN`	未设置	远程注册/注销应用及管理数据时的管理员校验令牌

15. 维护与帮助

日常升级与迁移前，请重点备份自定义应用包目录、外部数据集与模型权重、启动环境变量配置以及需长期留存的结果 JSON；运行期诊断日志统一保存在项目 `.run/` 目录下。遇到异常时，按 **“中枢是否可达 → 工作设备是否在线 → 节点依赖与数据是否齐全 → 失败节点具体报错文本”** 的顺序即可快速隔离网络、环境与业务逻辑问题。

更多排障解答见 常见问题、排障与维护，参与社区共建、访问 官方代码仓库 或在论文中引用请查阅 贡献与学术引用说明，许可边界见 开源许可与第三方说明（英文版手册见 English User Manual）。