工业视觉平台 ProcVision 算法SDK 设计意图（提案稿）

> 目的：形成项目团队共识，指导即将开展的实现工作（Phase 01）。本文以“将要实施”的视角阐述目标与方案，以便对齐接口、数据、工具链与交付标准。

背景与目标

- 背景：生产环境完全离线，平台与算法需解耦，算法以标准化包形式交付并热加载运行。
- 目标：
  - 建立统一的算法接口与返回值规范（两层结构）。
  - 单实例支持多 PID，依据 `pid` 运行时动态加载资源。
  - 统一数据传输路径（共享内存，JPEG-only），收敛 `image_meta` 字段。
  - 完整的 CLI 工具链支撑开发自测与离线打包。
  - 明确 Session 与错误码体系，强化健壮性与可运维性。

范围与不做的事

- 范围：Phase 01 面向单工位、单镜头、可重试流程，支持 OK/NG 判定与位置/缺陷框展示。
- 不做：
  - overlay 图像输出（Phase 01 不支持）。
  - 在线依赖分发与外网访问。
  - 复杂多进程/分布式推理编排。

关键设计决策

- 实例与 PID 解耦：
  - 构造函数不绑定 PID；在 `pre_execute/execute` 通过 `pid: str` 传入。
  - `manifest.json` 声明 `supported_pids` 与 `required_assets`；可配 `default_config` 兜底。
- 返回值两层结构（RESTful 两层）：
  - 首层：`status: OK|ERROR`、`message`（仅错误时）。
  - 业务层：`data.result_status: OK|NG`、`defect_rects<=20`、`position_rects`、`debug`。
- 统一时间：时间戳统一使用 `timestamp_ms`（Unix 毫秒）。
- 图像输入：共享内存传输 JPEG；`image_meta` 最小集合：`width,height,timestamp_ms,camera_id`。
- 参数类型系统：限定 `int,float,rect,enum,bool,string`，Dev Runner按 schema 校验。
- Session：仅在单次检测流程内有效，提供 `get/set/delete/exists`；移除 ttl/reset。
- 通信协议：stdin/stdout 管道，帧格式为 `4字节大端长度 + UTF-8 JSON`；心跳 `ping/pong` 保活。
- 标准错误码：统一 1001–1007、9999 等编码与含义，平台与 Runner 共识处理。

接口概要

- BaseAlgorithm 抽象：`get_info()`,`pre_execute(...)`,`execute(...)`；可选生命周期：`setup/teardown/on_step_start/on_step_finish/reset`。
- `get_info()` 返回：`name,version,description,supported_pids,steps[params...]`，与 `manifest.json` 一致。
- `pre_execute(...)`：前置条件检查/参考信息产出，不返回真实检测结果。

- `execute(...)`：核心检测与判定，`data.result_status` 为 OK/NG，NG 时给出 `ng_reason` 与缺陷/位置框。

数据结构与约束

- `defect_rects` 最大 20；`position_rects` 建议 ⩽ 20；`message` 建议 < 100 字；`ng_reason` 建议 < 50 字。
- ROI 与坐标必须在图像范围内；Dev Runner 对越界与数量进行校验/截断。

通信与心跳

- 握手：算法启动后输出 `{"type":"hello","sdk_version":"1.0"}`，Runner 回应 `hello` 完成握手。
- 心跳：Runner 每 5s 发送 `ping`，算法 2s 内回复 `pong`；超时按重试/终止策略处理。
- 调用：Runner 发送 `call`，算法返回 `result`；日志写入 stderr，协议帧走 stdout。

CLI 命令构想与行为

- validate
  - 用法：`procvision-cli validate [project] [--manifest <path>] [--zip <path>] [--json]`
  - 行为：校验项目结构与 `manifest.json`；输出人类可读清单或 JSON。
- run（Dev Runner）
  - 用法：`procvision-cli run <project> --pid <pid> --image <path> [--params <json>] [--json]`
  - 行为：模拟平台运行，校验 schema，展示预执行与执行结果、NG 原因与缺陷数。
- package（离线打包）
  - 用法：`procvision-cli package <project> [--output <zip>] [--requirements <path>] [--auto-freeze] [--wheels-platform <p>] [--python-version <v>] [--implementation <impl>] [--abi <abi>] [--skip-download]`

    - 行为：生成/使用 `requirements.txt`，按目标环境下载 wheels 并打包 `源码/manifest/requirements/assets/wheels` 到 zip。
    - 便捷参数：`-o/-r/-a/-w/-p/-i/-b/-s`，默认 `wheels-platform=win_amd64, python-version=3.10`。
- init（脚手架）
    - 用法：`procvision-cli init <name> [-d|--dir <dir>] [--pids <p1,p2>] [-v|--version <ver>] [-e|--desc <text>]`
    - 行为：生成算法项目骨架与 `manifest.json`，在 `main.py` 注释标注需修改位置（PID、步骤 schema、检测逻辑）。

CLI 输出约定

- 默认输出人类可读摘要与提示；如需机器可读，统一 `--json`。
- 关键结果示例：
    - validate：✅ `manifest.json 校验通过` / ❌ `缺少字段: supported_pids`
    - run：`预执行 OK；执行 NG（缺陷数: 3）`
    - package：`打包成功: <zip>` 或具体错误说明。

错误码与诊断

- 1001 invalid_pid；1002 image_load_failed；1003 model_not_found；1004 gpu_oom；1005 timeout；1006 invalid_params；1007 coordinate_invalid；9999 unknown_error。
- 诊断字段统一走 `debug`，用于性能与问题排查，不参与业务判定。

开发与交付流程

- 环境：独立虚拟环境；安装 SDK 与项目依赖。
- 源码结构：算法包源码目录 + `manifest.json` + `requirements.txt` + `assets/`（可选）+ `wheels/`。
- 离线依赖：`pip download -r requirements.txt -d ./wheels --platform <p> --python-version <v> --implementation <impl> --abi <abi>`。
- 打包：优先使用 `procvision-cli package`；命名规范 `[算法名]-v[版本]-offline.zip`。

验收标准（共识）

- 接口一致性：`get_info/pre_execute/execute` 行为与字段严格符合规范；`supported_pids` 与 `manifest.json` 完全一致。
- 数据约束：`defect_rects<=20`、坐标有效、`message/ng_reason` 长度与语义规范。
- 通信与心跳：hello 握手、ping/pong 保活、stdout/stderr 分流。
- CLI 工具链：四个子命令行为与参数完整实现；`--json` 输出一致。
- 离线交付：zip 内含源码、manifest、requirements、assets（如需）、wheels；可在目标工作站离线安装运行。
- 日志与诊断：结构化 logger（JSON 字段）、必要的 `debug` 指标。

风险与权衡

- 多 PID 配置管理复杂度↑：通过 `required_assets` 与 `default_config` 兜底控制风险。
- UI 性能与显示限制：严格控制缺陷数量与消息长度，必要时 Runner 截断。
- GPU/OOM 与长执行风险：设置超时与错误码，心跳保活+优雅终止。
- 离线依赖匹配：`platform/python/impl/abi` 需与目标环境严格一致，提供自动/手动两套打包路径。

工作拆分与交付物

- SDK 接口与返回值实现（含生命周期钩子与校验工具）。
- 通信协议与心跳实现（hello、ping/pong、call/result）。
- CLI 四子命令实现与统一输出协议（人类可读 / `--json`）。
- Dev Runner：共享内存模拟、Session 模拟、schema/返回值校验器。
- 打包链路：wheels 下载与压缩产物生成；zip 内结构校验。
- 文档与示例：最小可运行示例、参数类型系统示例、错误码用例。

> 期望：团队对以上接口、协议、CLI 工具链与交付标准达成一致，并以此为基线推进 Phase 01 的实现与验收。