AGENT SKILL SPECIFICATION

OpenSAI Usage Skill Specification

Translate the user's natural-language computing request into a controlled distributed execution using the connected `opensai_*` MCP tools, and return the domain output, measured duration, and key scheduling choices (the MCP connection itself is configured in the client beforehand).

Overview | Quickstart | User Manual | Agent Integration | FAQ

◈ Reader Guide (How This Document Is Used)

This specification corresponds to `skills/en/opensai/SKILL.md` inside the code archive—the instruction file loaded by MCP-capable AI agents (such as Claude, Cursor, or OpenClaw). Human developers can review it here to see how an agent maps natural-language goals into tool calls, idempotent polling, and metric explanations; for step-by-step server setup, start with the Agent Integration Guide.

1. Choose the Right Execution Entry Point

1. **Prefer registered applications**: Call `opensai_list_apps` first to inspect available applications, their `inputs`, tunable `knobs`, and default `qos`. When an application matches the task, invoke `opensai_run_app`.
2. **Fall back to DAGs or single tasks**: If no registered application fits, inspect available step implementations via `opensai_catalog` and compose a multi-step workflow (`opensai_submit_dag`), or call `opensai_submit_task` for a single algorithm. Query `opensai_list_devices` and `opensai_list_data` as needed and reuse discovery results while they remain valid.
3. **Match tool scope to user intent**: An explicit request to run or benchmark a task authorizes that execution—proceed directly without redundant confirmation prompts. When the user only asks to inspect cluster state or explain a past run, use read-only tools exclusively.

2. Map User Goals into Tool Arguments

User Intent	Tool Settings
Domain input values for this run (points, frame, scene)	Pass in `inputs`
Parameters that must stay locked for business correctness	Pin in `knobs`; leave remaining knobs open for platform adaptation
"Finish within N seconds" (meet deadline at highest feasible quality)	`mode: "deadline"`, `deadline_s: N`
"Finish as fast as possible" (with optional quality floor)	`mode: "fastest"`, plus optional `min_quality`
"Meet the deadline while minimizing resource usage"	`mode: "efficient"`, `deadline_s: N` (optional `min_quality`)

- **Argument layout**: MCP tools take `mode`, `deadline_s`, and `min_quality` as top-level fields, whereas the Python SDK nests them inside `qos={...}`. Unspecified fields inherit the application manifest's defaults. Always interpret quality scores on the scale declared by that specific application.

Standard `opensai_run_app` invocation when `cluster_analytics` is registered:

```json
{
  "app": "cluster_analytics",
  "inputs": {"points": 30000, "seed": 1},
  "deadline_s": 3,
  "wait": true,
  "wait_timeout": 60
}
```

3. Submit Once and Track Idempotently

- Record the returned `job_id` immediately and track completion via `opensai_wait_result` or inspect snapshots via `opensai_get_result`.
- A response with `done=false` means the client wait window elapsed while the job is still running—**continue polling the same `job_id` rather than submitting a duplicate job**.
- When retrying an uncertain submission after a transient transport error, reuse the same `idempotency_key` to prevent duplicate execution. On `RATE_LIMITED`, wait for `retry_after_s` before retrying.

4. Compose a DAG or Register an Application Package

- **DAG specification (`dag_spec`)**: Provide `name` and `nodes`, where each node defines a unique positive integer `id`, a catalog `algorithm`, domain `params`, and predecessor IDs in `before_nodes`. Always run `opensai_validate_dag` prior to submission. Specify resource requests in `resources` (`cores` as integer count, `mem_mb` and `storage_mb` in MiB, and note that `bw_mbps` currently takes **MB/s**); to pin a worker, pass its `allocation_index` as `direct_allocation_id`.
- **Application packages**: Combine an `app.yaml` manifest with self-contained `steps/Algorithm_*.py` modules exporting `__all__` and implementing `__init__(self, param_dict, data_dict)` and `run(self)`. OpenSAI injects `_artifact_in`, `_artifact_out`, `_alloc`, and `_data_dir` at runtime—never pass underscore-prefixed keys from the caller. Return domain output from `run()`, and report normalized quality (0–1) and numeric metrics via `self.quality` and `self.metrics`.
- **Safe registration**: Validate with `opensai_validate_app` before executing an explicitly authorized registration; ask for clarification only when an operation would overwrite an application from a different owner or expand scope beyond the user's request. Registration distributes step Python code only—third-party libraries, model weights, and large datasets must be prepared on eligible workers.

5. Verify Data Locality and Transfer Paths

- Check dataset locations with `opensai_list_data`. Declare strict data affinity via step `data` in manifests or `requires_data` in DAG nodes. Specify target worker indices in `opensai_upload_data` (`devices`), as omitting `devices` uploads to every online worker.
- In remote deployments, `path` arguments resolve on the **MCP server host**; supply `manifest` and `files` as inline text when registering remotely without host filesystem access.

- Cross-device artifacts move automatically over direct QUIC by default. Relay multipath is controlled at hub startup via `OPENSAI_MULTIPATH=1`; inspect job transfer path records to explain whether a run used direct or relay paths.

6. Report Actionable, Verifiable Results

1. **Lead with the outcome**: State job status (`SUCCEEDED` / `FAILED`) and domain output first, followed by wall-clock `duration_s`, measured quality, and key placement/resource/knob choices from `opensai_explain_job` (clearly distinguishing predicted estimates from measured outcomes).
2. **Interpret metrics accurately**: `duration_s` is end-to-end job wall time; `compute_s`, `fetch_s`, and `queue_s` sum across nodes and must not be added together when parallel branches overlap. `core_seconds` measures allocated CPU core-time, while `bytes_moved` counts useful payload bytes fetched (check path records for multi-hop relay traffic).
3. **Handle failures and long outputs**: On failure, inspect the failing node's error message—refresh discovery if an app/algorithm is missing, or check `opensai_list_data` for missing datasets. If output text is truncated, increase `max_result_chars` in `opensai_get_result` (up to `4000`) or save full results via the Python SDK.