

SAM 2.0 AGI - One-Page Summary (README-based)

What it is

SAM 2.0 is a hybrid Python/C multi-agent system with a web dashboard and slash-command interface. It combines a Python orchestrator with C-accelerated cores for meta-control and a dual-system simulation arena.

Who it's for

Not found in repo.

What it does

- Runs a unified Python orchestrator, API server, and CLI via `complete_sam_unified.py`.
- Serves a web dashboard and terminal UI (default at `http://localhost:5004`).
- Provides slash-command controls for agents and tasks such as `/help`, `/status`, `/research`, `/code`, `/finance`, `/websearch`.
- Uses C extensions for core subsystems: `sam_sav_dual_system`, `sam_meta_controller_c`, `multi_agent_orchestrator_c`, `specialized_agents_c`, and `consciousness_*`.
- Streams runtime events through JSONL logs and a live log panel (`logs/sam_runtime.jsonl`).
- Supports profile-based execution (full vs experimental) with profiles in `profiles/` and launch scripts.
- Persists state snapshots per profile and reloads on start (`sam_data//state.json`).

How it works (repo evidence)

- Entry point: `complete_sam_unified.py` boots the system, API server, and UI.
- Requests from UI/HTTP/SocketIO flow to the Python orchestrator, which calls C cores.
- C cores implement the dual-system arena, meta-controller, agent orchestration, and consciousness modules.
- Events are written to `logs/sam_runtime.jsonl`; snapshots saved under `sam_data//`.
- Optional model providers: local Ollama or hosted APIs configured by env keys.

How to run (minimal)

- `pip install -r requirements.txt`
- `python setup.py build_ext --inplace`
- Optional: install Ollama and pull a model (e.g., `ollama pull codellama:latest`).
- Start: `./run_sam.sh` or `python3 complete_sam_unified.py`

Requirements and profiles

- Python 3.10+ and a C compiler toolchain; training has separate `requirements_training.txt`.
- Profiles: full (kill switch enabled) and experimental (no kill switch) with `SAM_PROFILE` override.

Interfaces and commands

- Dashboard and terminal UI plus API endpoints such as `/api/health`, `/api/agents`, `/api/command`, `/api/terminal/execute`; groupchat via SocketIO.
- Slash commands include `/connect`, `/disconnect`, `/clone`, `/spawn`, `/revenue`, `/start`, `/stop`, `/clear`.

Security and remote access

- Login password, allowlists, session secret, and proxy trust configured in `.env.local`; optional Google and GitHub OAuth with `/api/oauth/help`.
- Remote access guidance: Tailscale for private access; Cloudflare Tunnel and Access for public URL.

Logging, state, finance

- Score handling marks missing score as pending and logs events; live log panel streams JSONL log.
- Finance summary endpoints: `/api/finance/summary`, `/api/revenue/metrics`, `/api/banking/metrics`.

Testing and checks

- Smoke tests for C extension imports; comprehensive tests via `SAM_TEST_MODE=1`; recursive checks via `tools/run_recursive_checks.sh`.
- Failure case simulation: `python3 ./simulate_failure_cases.py`.

Training and distillation

- Pipeline: distillation dataset build, training loop (LoRA or full), and regression gate; live groupchat distillation supported.
- Provider and regression environment overrides listed in README (policy, chat, two-phase boot, auto-connect, teacher pool).

Integrations and docs

- Revenue ops pipeline with approval and audit; Gmail OAuth integration; optional GitHub auto-backup to remotes.
- Docs: `DOCS/GOD_EQUATION.md`, `DOCS/ALIGNMENT.md`, and README-derived implementation specs; ChatGPT archive files listed in README.