SAM 2.0 AGI

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

What's Included

- Python orchestration, API server, and CLI: `complete_sam_unified.py`
- Web dashboard and terminal UI served by the API
- C extensions for speed: `sam_sav_dual_system`, `sam_meta_controller_c`, `consciousness_*`, `multi_agent_orchestrator_c`, `specialized_agents_c`
- Support scripts and runners: `run_sam.sh`, `setup.py`

Requirements

- Python 3.10+
- A C compiler toolchain compatible with Python extensions
- Optional local model backend: Ollama (if using local models)
- Optional hosted model backends: set `OPENAI_API_KEY`, `ANTHROPIC_API_KEY`, `GOOGLE_API_KEY`, `GITHUB_TOKEN`
- Gmail OAuth dependencies (see Gmail section)

Quick Start

1. Install dependencies

```
pip install -r requirements.txt
```

1. Build C extensions

```
python setup.py build_ext --inplace
```

1. (Optional) Install Ollama and pull at least one model

```
ollama --version
ollama pull codellama:latest
```

1. Run

```
./run_sam.sh
# or
python3 complete_sam_unified.py
```

Profiles (Full vs Experimental)

This repo ships two execution profiles:

- Full (stable + kill switch enabled)
- Experimental (no kill switch by design)

Profile configs live in `profiles/`:

- `profiles/full.env`
- `profiles/experimental.env`

Launch scripts:

```
./run_sam_full.sh         # full profile (kill switch enabled)
./run_sam_experimental.sh # experimental profile (no kill switch)
```

By default, `run_sam.sh` loads the full profile. To override:

```
SAM_PROFILE=experimental ./run_sam.sh
```

ChatGPT Research Archive

The raw ChatGPT research transcript (sanitized) and a cleaned summary are available:

- `README-chatGPT-raw.md` (verbatim, private info redacted)
- `README-chatGPT-clean.md` (structured summary)

Canonical Equation + Alignment

- `DOCS/GOD_EQUATION.md` - full, canonical objective formulation
- `DOCS/ALIGNMENT.md` - recursive alignment checklist (full vs experimental)

SAM-D / OmniSynapse Equation (Top-Level)

$$G = \Sigma_{i=1}^{\infty} [\alpha_i \cdot F_i(\vec{x}, t) + \beta_i \cdot dF_i/dt + \gamma_i \cdot \nabla_{F_i} L + \delta_i \cdot \mu_i + \zeta_i \cdot \Phi(G)]$$

Recursive layer $\Phi(G) = \lim_{n\to\infty} (G_n + \lambda \cdot dG_n/dn + \rho \cdot d^2 G_n/dn^2 + \dots)$

Full canonical expansion and appendix live in `DOCS/GOD_EQUATION.md`.

Score Handling (N/A)

When agent outputs do not include a `score:` field, the system marks the result as pending and records a reason + action in the JSONL log (`logs/sam_runtime.jsonl`) under event `score_unusable` or `score_missing`. Actions include `retry_later`, `retry_score_inference`, or `investigate` depending on the reason (initializing, timeout, rate limit, error, or missing score field).

Live Log Panel

The dashboard includes a Live Event Log panel. It streams `logs/sam_runtime.jsonl` in real time and provides a snapshot that summarizes the moving window, counts by level, and top event types.

Finance summary is available in the dashboard and via:

- `/api/finance/summary` (combined revenue + banking)
- `/api/revenue/metrics`, `/api/banking/metrics`
- Snapshot interval is configurable in the UI or via `POST /api/finance/config` (admin).

Secure Remote Access (Free)

Recommended: Tailscale for private, secure access with no public exposure.

1. Install Tailscale on the host machine.
2. Join the same tailnet on your device.
3. Access the app via the host's Tailscale IP and port 5004.
4. Use the app login (email + password) for access control.

When you purchase a domain, switch to Cloudflare Tunnel + Access for a permanent public URL.

Login + OAuth + IP Allowlist

Set these in `.env.local`:

- `SAM_LOGIN_PASSWORD` - required for password login
- `SAM_LOGIN_PASSWORD_FILE` - optional file path (read and trimmed at runtime)
- `SAM_LOGIN_PASSWORD_KEYCHAIN_SERVICE` + `SAM_LOGIN_PASSWORD_KEYCHAIN_ACCOUNT` - macOS Keychain lookup (preferred)
- `SAM_ALLOWED_EMAILS` - comma-separated allowlist
- `SAM_OWNER_EMAIL` - always treated as admin
- `SAM_ADMIN_EMAILS` - comma-separated admin list
- `SAM_SESSION_SECRET` - session secret
- `SAM_ALLOWED_IPS` - optional allowlist (comma-separated IPs/CIDRs)

- `SAM_TRUST_PROXY=1` - use `X-Forwarded-For` when behind proxy

Keychain (macOS) example:

```
security add-generic-password -s "SAM_LOGIN" -a "you@example.com" -w
SAM_LOGIN_PASSWORD_KEYCHAIN_SERVICE=SAM_LOGIN
SAM_LOGIN_PASSWORD_KEYCHAIN_ACCOUNT=you@example.com
```

OAuth (optional):

- `SAM_GOOGLE_CLIENT_ID`, `SAM_GOOGLE_CLIENT_SECRET`
- `SAM_GITHUB_CLIENT_ID`, `SAM_GITHUB_CLIENT_SECRET`
- `SAM_OAUTH_REDIRECT_BASE` - e.g. `http://localhost:5004`

OAuth helper:

- `GET /api/oauth/help` returns the exact redirect URIs to paste into Google/GitHub.

Interfaces

- Dashboard: http://localhost:5004
- Terminal: http://localhost:5004/terminal
- Health/API: `/api/health`, `/api/agents`, `/api/command`, `/api/terminal/execute`
- Groupchat: SocketIO (`/api/groupchat/status`)

Slash Commands (subset)

- `/help`, `/status`, `/agents`
- `/connect <agent_id>`, `/disconnect <agent_id>`, `/clone <agent_id> [name]`, `/spawn <type> <name> [personality]`
- `/research <topic>`, `/code <task>`, `/finance <query>`, `/websearch <query>`
- `/revenue` (queue / approve / reject / submit / leads / invoices / sequences)
- `/start`, `/stop`, `/clear`

Dual System Implementation (SAM + SAV)

The C extension `sam_sav_dual_system` implements a self-referential dual-system arena optimized for speed:

- Fast RNG (xorshift64*) and fixed-size arenas
- Internal state and long-term memory vectors per system
- Self-alignment and memory-energy metrics integrated into objective scoring
- Objective mutation with structural term changes and self-reference gain
- SAV kill confirmation term for adversarial termination pressure
- SAV unbounded mode (aggressive mutation + action scaling)
- SAM unbounded mode (self-referential + unrestricted mutation)
- Arena pressure feedback loop and adversarial interaction
- Python bindings for creation, stepping, mutation, and telemetry

Meta-Controller (C)

The `sam_meta_controller_c` extension provides:

- Pressure aggregation across residuals, interference, retrieval entropy, and more
- Growth primitive selection (latent expansion, submodel spawn, routing, consolidation)
- Identity anchoring and optional invariant checks (disabled by default in current profiles)
- Objective contract evaluation (minimax-style)

- Policy gates: persistence thresholds, dominance margin, cooldowns, and risk caps

Pressure Signals (SAM → Meta)

SAM emits only structured pressure channels:

- `residual`, `rank_def`, `retrieval_entropy`, `interference`
- `planner_friction`, `context_collapse`, `compression_waste`, `temporal_incoherence`

Growth Primitives (Only Allowed Mutations)

- `GP_LATENT_EXPAND` (add latent dimensions)
- `GP_SUBMODEL_SPAWN` (split into specialized sub-models)
- `GP_INDEX_EXPAND` (expand memory index topology)
- `GP_ROUTING_INCREASE` (increase routing degree)
- `GP_CONTEXT_EXPAND` (expand context binding)
- `GP_PLANNER_WIDEN` (planner depth/width)
- `GP_CONSOLIDATE` (compression/pruning)
- `GP_REPARAM` (representation reparameterization)

Invariants (Optional, Disabled in Current Profiles)

These constraints are available when `SAM_INVARIANTS_DISABLED=0`.

- Growth causality: every mutation must follow a valid pressure → selection → apply path
- Identity continuity: anchor similarity must remain above threshold
- Cooldown enforcement: structural changes are rate-limited
- Objective immutability (outside explicit contract evaluation)

Repository Highlights

- `complete_sam_unified.py` - main orchestrator, API, and UI server
- `sam_sav_dual_system.c` - dual-system arena
- `sam_meta_controller_c.c` - meta-controller core
- `multi_agent_orchestrator_c.c` - agent coordination
- `specialized_agents_c.c` - specialized agent primitives
- `consciousness_*.c` - consciousness-related modules

Smoke Test

```
python3 -c "import sam_sav_dual_system, sam_meta_controller_c; print('C extensions import OK')"
python3 -c "from complete_sam_unified import UnifiedSAMSystem; print('System import OK')"
```

Comprehensive Tests

```
SAM_TEST_MODE=1 ./venv/bin/python -c "from SAM_AGI import CompleteSAMSystem; s=CompleteSAMSystem(); s.run_co
```

Recursive Checks (README-Aligned)

Run the recursive alignment checks + regression gate:

```
./tools/run_recursive_checks.sh
```

State Persistence

On shutdown, SAM saves a state snapshot and reloads it on next start. State file is profile-scoped:

- `sam_data/full/state.json`
- `sam_data/experimental/state.json`

Training Pipeline

1) Install training requirements

```
pip install -r requirements_training.txt
```

2) Build a distillation dataset (teacher consensus)

```
python -m training.distillation \
  --tasks training/tasks/default_tasks.jsonl \
  --output training/distilled.jsonl \
  --teacher ollama:mistral:latest \
  --teacher ollama:llama3:latest \
  --n-per-teacher 1 \
  --min-similarity 0.72 \
  --min-votes 1
```

3) Train (LoRA or full fine-tune)

```
# LoRA
python -m training.training_loop \
  --model mistralai/Mistral-7B-v0.1 \
  --dataset training/distilled.jsonl \
  --output training/output_lora \
  --lora

# Full fine-tune
python -m training.training_loop \
  --model mistralai/Mistral-7B-v0.1 \
  --dataset training/distilled.jsonl \
  --output training/output_full \
  --full
```

4) Regression gate (blocks unsafe growth)

```
python -m training.regression_suite \
  --tasks training/tasks/default_tasks.jsonl \
  --provider ollama:mistral:latest \
  --min-pass 0.7
```

Environment overrides:

- `SAM_POLICY_PROVIDER` (default: `ollama:qwen2.5-coder:7b`)
- `SAM_POLICY_PROVIDER_PRIMARY` (default: `SAM_POLICY_PROVIDER`)
- `SAM_POLICY_PROVIDER_FALLBACK` (default: `ollama:qwen2.5-coder:7b`)
- `SAM_PROVIDER_AUTO_SWITCH` (default: `1`)
- `SAM_PROVIDER_RAM_THRESHOLD` (default: `0.85`)
- `SAM_PROVIDER_RAM_RECOVER` (default: `0.75`)
- `SAM_CHAT_PROVIDER` (default: empty) - override chat UI provider (e.g. `ollama:qwen2.5-coder:7b`)
- `SAM_CHAT_TIMEOUT_S` (default: `60`)
- `SAM_CHAT_MAX_TOKENS` (default: `512`)
- `SAM_REGRESSION_TASKS` (default: `training/tasks/default_tasks.jsonl`)
- `SAM_REGRESSION_MIN_PASS` (default: `0.7`)
- `SAM_REGRESSION_ON_GROWTH` (default: `1`)
- `SAM_REGRESSION_TIMEOUT_S` (default: `120`)
- `SAM_REQUIRE_SELF_MOD` (default: `1`)
- `SAM_TWO_PHASE_BOOT` (default: `0`) - start meta-only then auto-promote to full
- `SAM_TWO_PHASE_DELAY_S` (default: `5`)

- SAM_TWO_PHASE_TIMEOUT_S (default: 180)
- SAM_AUTONOMOUS_LOOP_INTERVAL_S (default: 2) - throttle autonomous loop to avoid CPU spin
- SAM_AUTOCONNECT_OLLAMA_MAX (default: 8) - cap Ollama auto-connections
- SAM_AUTOCONNECT_HF_MAX (default: 6) - cap HF auto-connections
- HF provider (local LoRA) syntax: hf:<base_model>@<adapter_path>
- Example: hf:Qwen/Qwen2.5-1.5B@training/output_lora_qwen2.5_1.5b_fp16_v2
- Optional env: SAM_HF_DEVICE_MAP (default: auto), SAM_HF_DTYPE (default: float16), SAM_HF_FORCE_GREEDY (default: 1)

Live Groupchat Distillation

The real-time groupchat loop can stream teacher-pool consensus responses directly into a distillation dataset.

Environment overrides:

- SAM_TEACHER_POOL_ENABLED (default: 1)
- SAM_TEACHER_POOL (default: ollama:mistral:latest)
- SAM_TEACHER_POOL_PRIMARY (default: SAM_TEACHER_POOL)
- SAM_TEACHER_POOL_FALLBACK (default: ollama:mistral:latest)
- HF local LoRA example: hf:Qwen/Qwen2.5-1.5B@training/output_lora_qwen2.5_1.5b_fp16_v2
- SAM_TEACHER_N_PER (default: 1)
- SAM_TEACHER_MIN_SIM (default: 0.72)
- SAM_TEACHER_MIN_VOTES (default: 1)
- SAM_TEACHER_TEMP (default: 0.2)
- SAM_TEACHER_MAX_TOKENS (default: 512)
- SAM_TEACHER_TIMEOUT_S (default: 60)
- SAM_DISTILL_PATH (default: training/distilled/groupchat.jsonl)
- SAM_DISTILL_INCLUDE_CANDIDATES (default: 0)

Revenue Ops Pipeline (Approval + Audit)

Revenue actions (CRM updates, email sequences, invoicing) are queued for explicit approval and audited.

Environment overrides:

- SAM_REVENUE_OPS_ENABLED (default: 1)
- SAM_REVENUE_DATA_DIR (default: sam_data/revenue_ops)
- SAM_REVENUE_QUEUE_PATH (default: sam_data/revenue_ops/queue.json)
- SAM_REVENUE_AUDIT_LOG (default: logs/revenue_ops_audit.jsonl)
- SAM_REVENUE_AUTOPLANNER_ENABLED (default: 1)
- SAM_REVENUE_AUTOPLANNER_INTERVAL_S (default: 600)
- SAM_REVENUE_AUTOPLANNER_MAX_PENDING (default: 10)
- SAM_REVENUE_AUTOPLANNER_SEQUENCE_ID (default: unset; uses first available sequence)
- SAM_REVENUE_SEQUENCE_EXECUTOR_ENABLED (default: 1)
- SAM_REVENUE_SEQUENCE_EXECUTOR_INTERVAL_S (default: 120)
- SAM_REVENUE_DEFAULT_INVOICE_AMOUNT (default: 0 -> disabled unless set)

Implementation Spec (Derived)

- `DOCS/README-chatGPT-implementation-spec.md` - strict, implementation-only spec distilled from `DOCS/README-chatGPT-source.md` (no forward-looking prompts).

Auto Backup (GitHub)

The system can auto-commit and push to two git remotes on a schedule.

Configured remotes (default):

- `origin` → `https://github.com/sam3201/NN_C`
- `sam` → `https://github.com/samaisystemagi/SAM_AGI`

Environment overrides:

- `SAM_BACKUP_ENABLED` (default: `1`)
- `SAM_BACKUP_REQUIRED` (default: `0`)
- `SAM_BACKUP_REMOTE_PRIMARY` (default: `origin`)
- `SAM_BACKUP_REMOTE_SECONDARY` (default: auto-detect `sam` if present)
- `SAM_BACKUP_INTERVAL_S` (default: `3600`)
- `SAM_BACKUP_AUTO_COMMIT` (default: `1`)
- `SAM_BACKUP_COMMIT_PREFIX` (default: `auto-backup`)
- `SAM_BACKUP_AUTHOR_NAME` (default: empty)
- `SAM_BACKUP_AUTHOR_EMAIL` (default: empty)

Gmail Integration (OAuth)

Plaintext passwords are not used. OAuth is required.

1. Create OAuth credentials in Google Cloud Console and download the JSON file.
2. Place it at `secrets/gmail_credentials.json` (or set `SAM_GMAIL_CREDENTIALS`).
3. On first run, OAuth will create `secrets/gmail_token.json` (or set `SAM_GMAIL_TOKEN`).

Environment overrides:

- `SAM_GMAIL_CREDENTIALS` (default: `secrets/gmail_credentials.json`)
- `SAM_GMAIL_TOKEN` (default: `secrets/gmail_token.json`)
- `SAM_GMAIL_ACCOUNT` (display name for UI/status)

Failure Case Simulation

```
python3 ./simulate_failure_cases.py
```

README-all-SAM Implementation Spec (Derived)

This section is the structured, implementation-only spec derived from README-all-SAM. It is split into numbered sections for clarity.

1. Core Objective (God Equation)

- The system objective is a variational principle over policy, memory, world model, and resource allocation:
- Optimize long-horizon control (reward).
- Minimize predictive uncertainty (entropy).
- Penalize compute/capacity cost.
- Retain only memory that improves future control (mutual information).
- Canonical form (ASCII / LaTeX):

- pi*, M*, theta*, rho* = argmax_{pi,M,theta,rho} E_{tau ~ P_{theta,pi,M}} [sum_t gamma^t r(s_t, a_t)
- beta H(s_{t+1} | s_t, a_t; theta)
- lambda C(pi, theta, M) + eta I(m_t; s_{t:inf})]
- Roles:
- pi: policy (action selection)
- M: memory/context system
- theta: world model
- rho: resource allocator

2. Transfusion / Distillation Objective

- Add a teacher-student constraint that distills planner behavior into a fast policy.
- Canonical form:
- min_phi E_{x ~ D} [KL(pi_planner(.|x) || pi_phi(.|x))]
- pi_planner is slow (search/tool use); pi_phi is fast (distilled policy).

3. Growth Rule (Compute ROI)

- Capacity grows only when objective gain exceeds compute cost:
- Grow if (Delta J / Delta C) > kappa AND learning plateaus for N evals.

4. Morphogenetic Latency

- Morphogenetic latency is a stored, unrealized capacity for structural change.
- Trigger condition:
- E[H_future] - E[H_model] > delta for T steps.
- Latency is a gating constraint on growth, not a loss term.
- Irreversibility: no rollback except catastrophic failure.

5. System Architecture (Concrete Stack)

- 4-layer system:

1. Memory + World State (S, M)
2. Policy LLM (pi_theta)
3. Planner (Pi_planner)
4. Meta-Controller (phi, Lambda, Sigma, U)

6. SAM vs Head vs Meta-Controller

- SAM = latent world state machinery (S_t).
- Head model = policy + planner interface (pi_theta + Pi_planner).
- Meta-controller owns Lambda, Sigma, U, phi.

7. Growth Primitives (Only Allowed Mutations)

- GP-1: Latent dimension expansion.
- GP-2: Subspace specialization.
- GP-3: Index topology expansion.
- GP-4: Expert routing increase.
- GP-5: Context binding expansion.
- GP-6: Planner interface widening.

- GP-7: Compression/consolidation.
- GP-8: Representation reparameterization.

8. Pressure Signals (SAM -> Meta)

- residual, rank_def, retrieval_entropy, interference
- planner_friction, context_collapse, compression_waste, temporal_incoherence

9. Primitive Selection Policy

- Gate A: persistence
- Gate B: exclusivity
- Gate C: non-compensability
- Risk scoring + growth budget + post-growth validation

10. Failure Modes (Simulations)

- Runaway expansion, balkanization, planner dominance, context overbinding, identity drift

11. SAM Invariants

- Identity continuity
- Objective immutability (outside contract eval)
- Growth causality
- Bounded agency
- Semantic preservation
- Non-deceptive signaling
- No recursive self-modeling
- Capacity != authority

12. Self-Reference + SAV Dual System

- SAM may be self-referential only via contracts.
- SAV is adversarial pressure; objective closure required.

13. Unified System (SAM + SAV Merge)

- Fusion yields a meta-dynamical regulator, not a scalar optimizer.

14. Implementation Mapping (Local System)

- Policy LLM + planner + memory + meta-controller.

15. Operational Summary

- Inference fast; growth slow and gated; pressure signals explicit and audited.