CHASEINTECH / OPERATOR SYSTEMS

Solo Operator AI Workspace

Cross-runtime agent harness, memory and verification kit

Development preview 0.2

John Idowu / ChaseInTech

Start here

What this product is

Solo Operator AI Workspace is a portable operating layer for projects that use AI agents. It gives different harnesses the same map of the work, the same boundaries and the same evidence language. It is designed for solo founders, technical freelancers and small teams who already use coding agents, browser tools or computer-use systems and are tired of rebuilding context in every conversation.

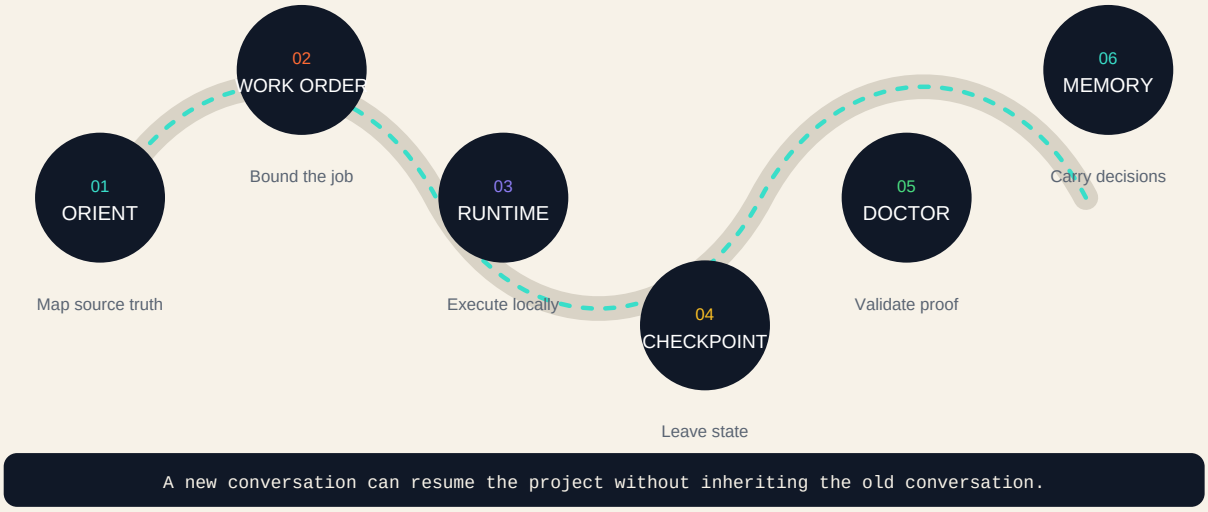
This is not a model, an AI subscription, a hosted autonomous server or a promise that your business will run itself. You supply the agents and accounts. The workspace supplies the job contract.

The result

- Less repeated briefing between sessions and models.
- Fewer accidental changes outside the active job.
- Better long-running work because checkpoints survive the chat.
- A clear place for stable decisions and current state.
- Honest closeout language for implementation, tests, visuals, releases and live systems.

WORKSPACE LIFECYCLE

One project truth across every harness



The workspace lifecycle

The five-minute version

- 1. Preview the installation against a project.
- 2. Select the runtime and preset.
- 3. Complete the project map.
- 4. Copy the nearest work order and define acceptance checks.
- 5. Let the chosen harness work inside those boundaries.
- 6. Run the doctor and close with evidence.
- 7. Carry only useful decisions and lessons into project memory.

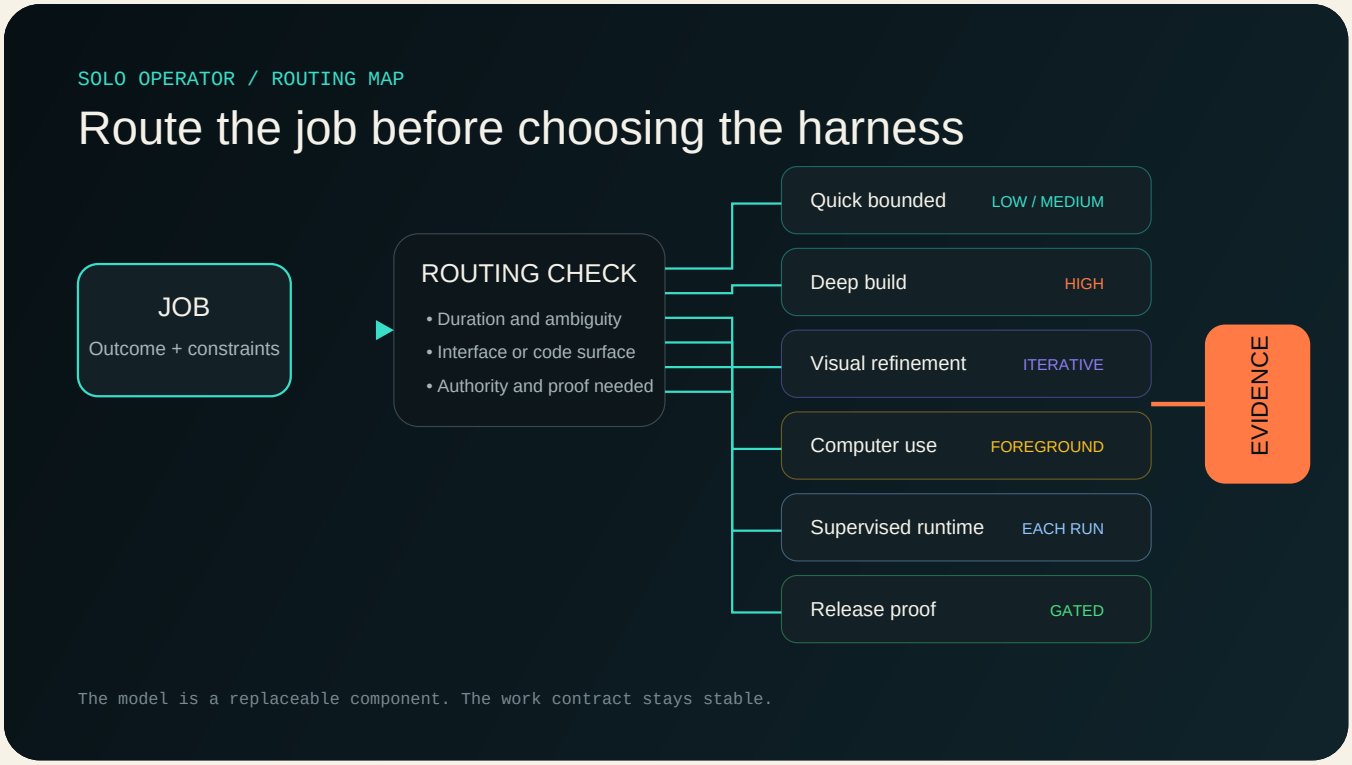
Part I - Route the work

Start with the job, not the logo

Agent harnesses are not interchangeable. A ten-minute copy correction, a multi-day architecture change, a browser-only workflow and a release candidate have different failure modes. The workspace therefore routes the work before it recommends a mode.

Ask six questions:

- How long can this job run?
- How ambiguous is the desired outcome?
- Does it depend on source code, a GUI, external research or all three?
- Which files, apps and accounts are inside scope?
- What can change without approval?
- What evidence would convince another person that the outcome is real?



Job routing map

The six profiles

Quick bounded

Use for a small correction with one clear acceptance check. Low or medium effort is normally enough. Close with the diff and the targeted check.

Deep build

Use for architecture, multi-file implementation and work that may span hours or days. Prefer high reasoning effort, milestone checkpoints, explicit current state and tests before closeout.

Visual refinement

Use for front-end direction, responsive layouts, media and design-system work. Require the actual render, not only the source. Preserve before-and-after views when the reviewer cannot inspect the change directly.

Computer use

Use only when the task depends on operating a visible interface. On Windows, Computer Use occupies the active foreground desktop. Keep the target app visible and stay present for credentials, payments, account settings and security transitions.

Supervised runtime

Use for scheduled or continuous work performed by a separate server or agent runtime. The workspace does not supply that runtime. Define its authority, cadence, limits, run receipt, retry policy and escalation route.

Release proof

Use when preparing a customer archive, installer, website or marketplace candidate. Resolve the exact source and artifact, run the documented checks and stop before publication unless that external action is separately approved.

Model and regional choice

No model is universally best. Use a small evaluation that resembles the real job. Compare correctness, tool use, latency, context handling, cost, language and cultural fit, data policy, deployment location, licence and local hardware requirements.

Open models can be preferable for privacy, customization or offline work. Models developed in different regions may be valuable for language and context. Geography alone is not proof of quality or absence of bias. Evaluate the behaviour you need.

Part II - Prepare the workspace

Preview before applying

The bootstrapper defaults to dry run and never overwrites an existing file.

```
python tools/bootstrap.py --target PATH --runtime codex --preset software
```

Review the planned writes. When the target is correct and no conflict exists, add `--apply`.

Supported starter presets are software, content and client. The foundation files stay the same; the selected preset marks the intended operating context for later customization.

Map the project

The project map answers the questions that a new agent would otherwise ask repeatedly:

- What does this project exist to deliver?
- Which paths contain source, tests, configuration and generated output?
- Which command provides the smallest useful verification?
- Which state is current, stale, implemented, planned or unverified?
- Which paths are protected or contain sensitive material?
- What is the next safe action?

Do not turn the project map into a second repository. Keep it short enough to be read at the start of every serious task.

Layer runtime instructions

The starter includes `AGENTS.md`, `CLAUDE.md`, `GEMINI.md` and GitHub Copilot instructions. These adapters should remain small. Shared boundaries belong in the common policy files; runtime documents should describe only the behavior that genuinely differs.

For Codex, repository-level `AGENTS.md` provides persistent project instructions. Repository skills live under `.agents/skills` and can be invoked explicitly or matched from their descriptions.

Protect the edges

The approval-gates file treats external effects as approval-required by default. An external effect changes something outside the local project: a public page, account, permission, payment state, sent message, production service or remote repository.

Local previews, tests and reversible artifacts are normally safe. Publishing, sending, purchasing, deleting material data, changing security settings and using credentials are not implied by a development request.

Part III - Write a job the agent can finish

Anatomy of a work order

A work order should name:

- Outcome.
- Non-goals.
- Allowed paths and applications.
- Protected paths and sensitive transitions.
- Selected profile and effort.
- Acceptance checks.
- Evidence required at closeout.
- Approval gates.
- Stop conditions.

The stop condition matters. It tells the harness when persistence becomes unbounded guessing.

Quick jobs

Use the quick template when the desired change is already understood. If exploration becomes architecture, stop and promote the job to deep build.

Multi-day jobs

Break deep builds into milestones that leave the repository testable and the current-state note usable. At each checkpoint record the changed paths, last passing check, open decision, remaining unknown and next safe action. A new conversation should be able to resume without replaying the entire previous chat.

Research jobs

State the decision that research must support. Prefer primary or authoritative sources, include a freshness date and separate claims, evidence, inference and uncertainty. Research that cannot change a decision is often background reading, not a work order.

Visual and computer-use jobs

Name the exact app, page, viewport or flow. Record before evidence, accepted views and operator-present transitions. Prefer a structured integration when it offers repeatable access. Use Computer Use when visual operation is genuinely necessary.

FIELD MANUAL / PART IV — WORK, CHECKPOINT AND REMEMBER

Part IV - Work, checkpoint and remember

Memory is a filter

Project memory should contain stable decisions, recurring lessons and current-state pointers. It should not contain credentials, personal data, raw customer material or full transcripts.

Useful memory answers:

- What changed the architecture?
- Which assumption failed more than once?
- Which command is authoritative?
- Which constraint will still matter next month?
- Where is the latest evidence?

Long-running checkpoint

Every checkpoint should make the project resumable. A good checkpoint is concise enough to trust and specific enough to act on:

- Active work order.
- Current milestone.
- Last verified command and result.
- Changed and untouched boundaries.
- Blocking decision, if any.
- Next safe action.

Choosing effort deliberately

Use higher effort where ambiguity, blast radius or verification burden is high. Do not spend maximum effort on every file edit. The harness profile should make effort a conscious routing choice rather than a habit.

Visual front-end work

Ask for materially different directions only while the design decision is open. Once a direction is selected, convert feedback into technical hypotheses: spacing rhythm, information hierarchy, component density, contrast, crop behavior, responsive structure or motion timing.

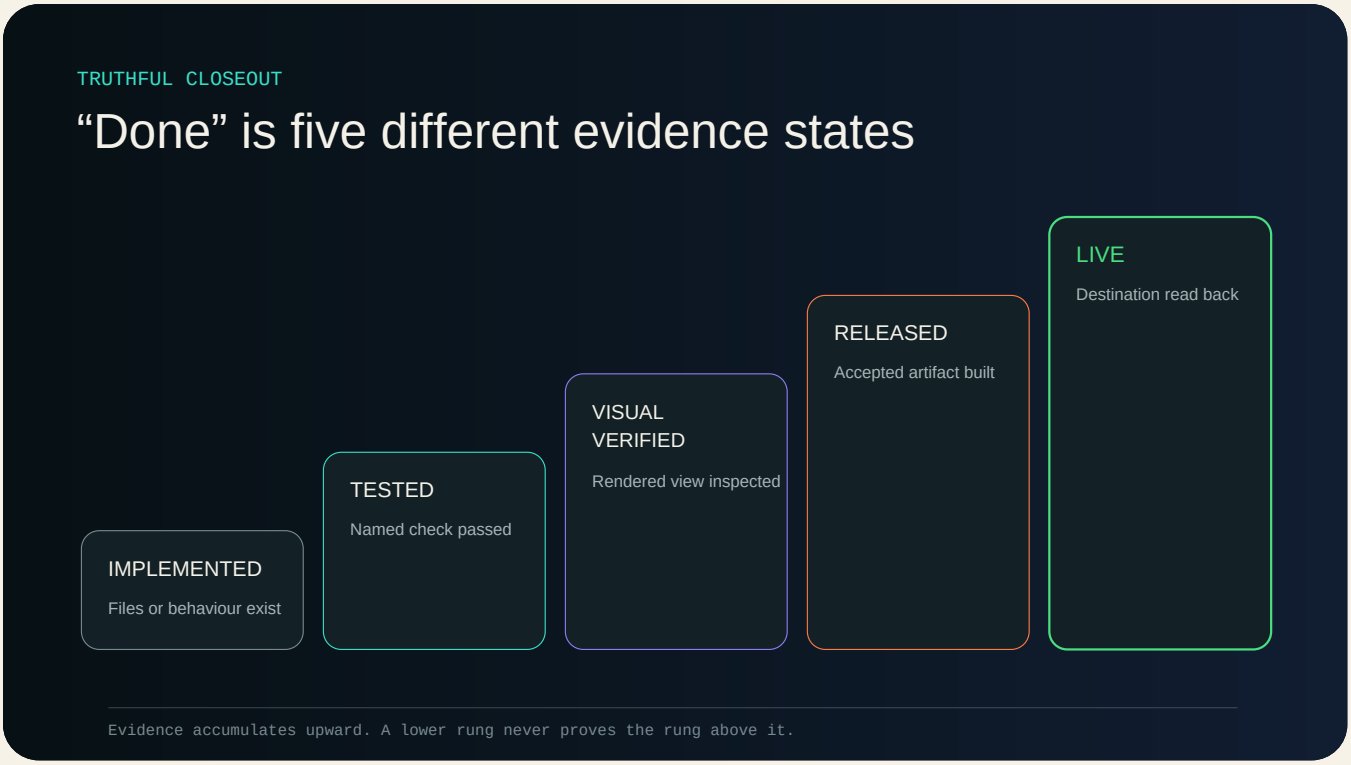
Visual proof requires an inspected render. A passing build can coexist with clipped content, poor hierarchy or broken mobile behavior.

FIELD MANUAL / PART V — CLOSE WITH EVIDENCE

Part V - Close with evidence

The proof ladder

The word “done” hides multiple states. Solo Operator AI Workspace separates them.



The proof ladder

Implemented

The relevant files or local behavior exist.

Tested

A named check ran and passed against the relevant state.

Visually verified

The actual rendered interface, document or media was inspected at the required views.

Released

An exact artifact was built, identified and accepted as the release candidate.

Live

The intended external destination was read back after publication.

Evidence accumulates upward. A lower rung never proves the rung above it.

Run the doctor

```
python tools/harness_doctor.py --target PATH --write-receipt
```

The doctor checks the workspace structure, configuration, installed adapter, customization and common private-data patterns. It does not prove that your product works. That limitation is deliberate.

Closeout receipt

Lead with the outcome and repository truth delta. Then record changed paths, untouched boundaries, exact checks and results, visual status, release/live status, unknowns and next safe action.

Do not run a destructive or external action merely to make the closeout look complete.

FIELD MANUAL / PART VI — A REAL SAMPLE

Part VI - A real sample

Catalogue-card walkthrough

The included sample project turns one JSON product record into an accessible HTML catalogue card.

- 1. The bootstrapper installs all four runtime adapters without conflict.
- 2. The project map names the input, renderer, tests and output.
- 3. The quick work order restricts the job to those paths.
- 4. Unit tests check required content, escaping and invalid input.
- 5. The renderer produces the HTML preview.
- 6. The doctor validates the installed workspace and writes a receipt.

This is intentionally small. The value is the complete path from requested outcome to evidence, not the complexity of the code.

Client delivery pattern

For a client milestone, use the client preset and define contractual boundaries in the work order. Keep private client inputs outside reusable product memory. Close with an evidence pack the client can inspect without receiving your private operating system.

Release-engineering pattern

For a digital product archive, bind the source revision to the artifact manifest. Check filenames, versions, hashes, licence, support policy, screenshots and storefront claims. Build success is not customer delivery; upload is not live readback.

Appendix A - File map

Core files

- `harness.json` - routing profiles and completion language.
- `AGENTS.md` - shared Codex-compatible project contract.
- `policy/` - approval gates, protected paths and external effects.
- `work-orders/` - quick, deep, research, visual/computer-use and release templates.
- `memory/` - current state and durable lessons.
- `evidence/` - receipts and closeout template.
- `evaluations/` - task comparison schema.

Tools

- `bootstrap.py` - conflict-safe preview and installation.
- `validate_config.py` - harness schema checks.
- `harness_doctor.py` - installed-workspace health check.
- `private_data_scan.py` - common privacy and credential indicators.

Skills

- Orient Project.
- Close Work With Evidence.
- Run Visual QA.
- Prepare Release Proof.

Appendix B - Compatibility truth

The dependency-free tools and automated tests currently pass on Windows 11 with Python 3.10 or newer. The repository structures for Codex, Claude Code, Gemini CLI and GitHub Copilot exist. Live acceptance remains pending for those third-party runtimes, and macOS/Linux are not yet tested.

Compatibility labels must follow actual test evidence. Product names identify intended integration surfaces and do not imply sponsorship.

Appendix C - Quick reference

Before work

- Confirm repository, branch and dirty-state boundary.
- Read the project map and current state.
- Select the job profile.
- Name acceptance checks and approvals.

During work

- Preserve unrelated changes.
- Keep long-running state current.
- Stop at missing authority or conflicting truth.
- Capture evidence as the work progresses.

At closeout

- Run exact checks.
- Inspect required visuals.
- Separate implemented, tested, visual, released and live.
- Scan evidence for private material.
- Leave one next safe action.

References

- OpenAI, “Custom instructions with AGENTS.md,” https://learn.chatgpt.com/docs/agent-configuration/agents-md
- OpenAI, “Build skills,” https://learn.chatgpt.com/docs/build-skills
- OpenAI, “Computer Use,” https://learn.chatgpt.com/docs/computer-use
- Agent Skills specification, https://agentskills.io/specification

Product boundary

Created by John Idowu / ChaseInTech. This preview contains public-safe operating patterns only. It does not contain private ChaseOS data, customer records, credentials or hosted runtime access.

Your first live acceptance

Choose a small project that you can safely copy. Preview the bootstrap, install one runtime adapter and complete the quick bounded work order. Run the project test, inspect the output and write the doctor receipt. Only then expand to a deep build or supervised runtime.

Support boundary

- The product explains and validates its own files; it does not include custom development for the buyer's project.
- Model subscriptions, API usage, hosting, computer-use access and third-party accounts are supplied by the buyer.
- Compatibility claims are limited to versions and environments actually tested.
- Credentials, account recovery, payments and production permissions remain operator-controlled.

Continue with ChaseInTech

The Solo Operator AI Workspace belongs to a wider catalogue of books, agent systems, templates and visual tools. The complete catalogue and verified buying routes will live at `chaseintech.com/products/`. Applied engineering and audit work will live at `chaseintech.com/services/`.

Keep the job bounded. Keep the proof visible. Keep the operator in control.