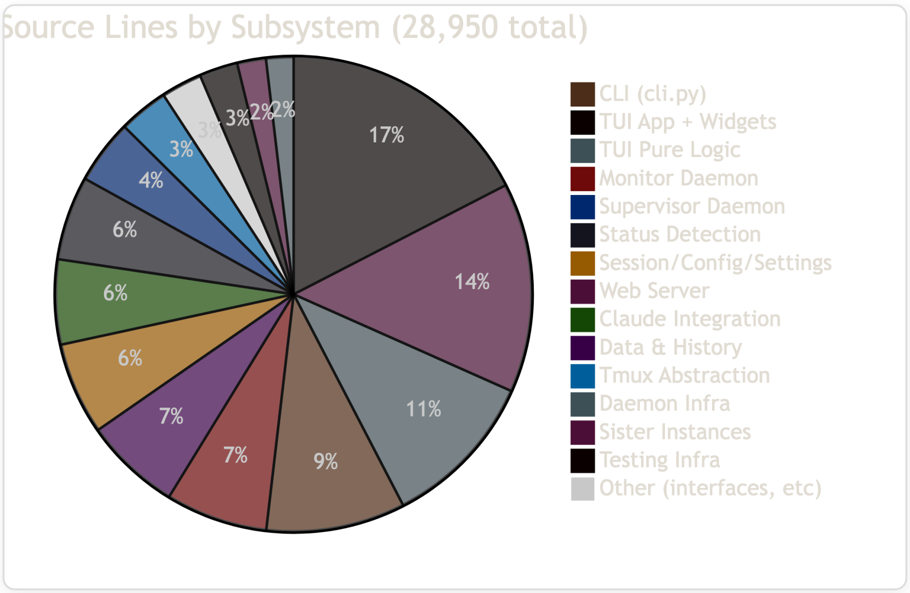
Overcode Architecture

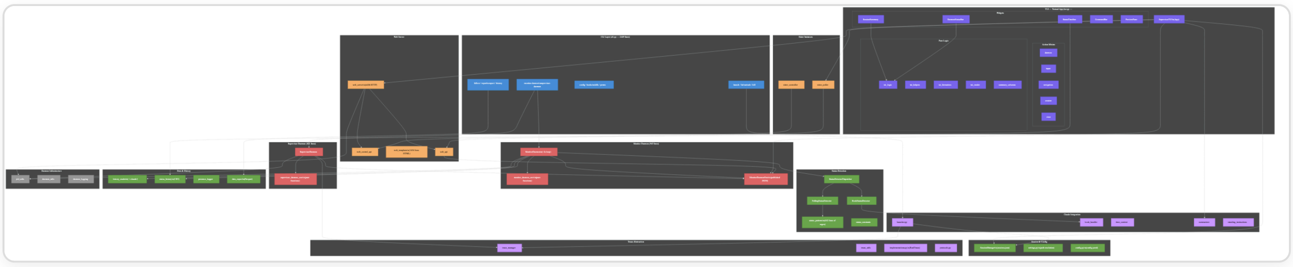
System Design & Improvement Roadmap

February 2026



Source lines by subsystem (28,950 total)

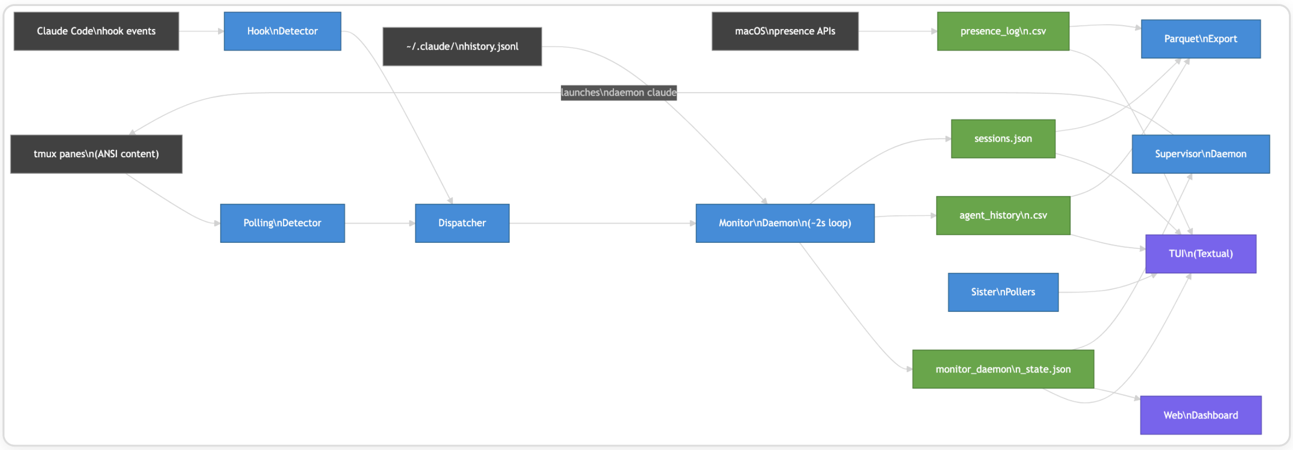
System Architecture



Module dependency graph — arrows show import/data flow direction

Data Flow

The core data pipeline is simple: tmux panes are scraped for status, the monitor daemon aggregates everything into a single JSON file, and all UIs read that file.



Runtime data flow

Subsystem Breakdown

CLI — 2,449 lines

Single file (`cli.py`) using Typer with 7 sub-command groups and ~25 top-level commands. All heavy imports are lazy (inside command functions) to keep startup fast.

Size concern: At 2,449 lines, this is the largest single file. It's essentially a dispatch table — each command function does minimal work before delegating to a library module.

TUI — 7,304 lines (25% of codebase)

The largest subsystem, split across:

Component	Lines	Role
`tui.py`	2,159	Main Textual App, refresh loop, composition
Widget files (8)	2,361	SessionSummary, CommandBar, DaemonStatusBar, etc.
Action mixins (5)	1,624	Keyboard handlers split by domain
Pure logic (4)	2,784	`tui_logic` , `tui_helpers` , `tui_formatters` , `tui_render`
`summary_columns`	812	Column registry with render functions

The TUI already has good separation between pure logic and UI code. The action mixin pattern keeps `tui.py` from being even larger, though the mixins themselves are tightly coupled to `SupervisorTUI` 's internal state.

Status Detection — 1,481 lines

Four modules implementing a dual-strategy pattern:

- **Polling detector** (457 lines): scrapes tmux pane content with 442 lines of regex patterns. This is the oldest and most battle-tested code.
- **Hook detector** (251 lines): reads Claude Code hook state files. Authoritative when fresh (<120s), falls back to polling.
- **Dispatcher** (83 lines): fan-out to both, selects best.
- **Constants** (248 lines): status strings, color maps, emoji maps.

Monitor Daemon — 1,795 lines

- `monitor_daemon.py` (943 lines): the main event loop
- `monitor_daemon_state.py` (459 lines): the published data model
- `monitor_daemon_core.py` (393 lines): pure business logic (time accumulation, cost calculation, heartbeat eligibility)

Supervisor Daemon — 1,135 lines

- `supervisor_daemon.py` (821 lines): reads monitor state, decides when to intervene, launches "daemon claude" in a tmux window, monitors completion
- `supervisor_daemon_core.py` (314 lines): pure logic for context building, session filtering, action determination

Web Server — 3,699 lines (13% of codebase)

- `web_templates.py` (1,656 lines): **the single biggest contributor to bloat** — entire HTML/CSS/JS dashboard as Python string literals
- `web_server.py` (642 lines): stdlib `http.server` routing
- `web_api.py` (844 lines): data aggregation for JSON API endpoints
- `web_control_api.py` (525 lines): agent control actions
- `web_chartjs.py` (32 lines): bundled Chart.js

Session & Config — 1,627 lines

- `session_manager.py` (834 lines): CRUD for the per-tmux-session registry
- `settings.py` (493 lines): all path resolution, dataclasses for daemon/TUI/presence settings
- `config.py` (300 lines): `~/.overcode/config.yaml` reader/writer

Claude Integration — 1,498 lines

- `launcher.py` (524 lines): creates tmux windows, starts Claude Code processes
- `standing_instructions.py` (285 lines): instruction presets
- `time_context.py` (315 lines): generates the clock/presence line for hooks
- `summarizer_component.py` + `summarizer_client.py` (474 lines): LLM-powered activity summaries
- `hook_handler.py` (123 lines): processes Claude Code hook events

Data & History — 1,704 lines

- `history_reader.py` (685 lines): reads Claude Code's JSONL history for token/cost data
- `status_history.py` (308 lines): per-agent status CSV logging
- `presence_logger.py` (454 lines): macOS user presence sampling
- `data_export.py` (257 lines): Parquet export for Jupyter

Architectural Assessment

What Works Well

1. **The daemon state file pattern.** Having the monitor daemon publish a single JSON file that all consumers read is clean and simple. No IPC, no message queues, no shared memory. Consumers are fully decoupled from the detection/aggregation loop.
2. **Pure logic extraction.** The `*_core.py` and `tui_logic.py` modules make business logic independently testable without mocking tmux/Textual/filesystem.
3. **Lazy imports in CLI.** Keeps `overcode --help` fast (~100ms) despite the large import graph.
4. **The dual status detection strategy.** Hook-based detection is authoritative and instant; polling is the reliable fallback. The dispatcher pattern makes this transparent to consumers.

What's Concerning

1. Size: 29K Lines for What It Does

Overcode's core job is: **watch tmux panes, report status, display a dashboard**. The 29K line count suggests significant accidental complexity. The biggest contributors:

Category	Lines	%	Assessment
Web templates (HTML/CSS/JS)	1,656	5.7%	Should be external files
Status patterns (regex)	442	1.5%	Inherent complexity
Summary columns (render fns)	812	2.8%	Could be data-driven
CLI dispatch	2,449	8.5%	Mostly boilerplate
TUI actions (5 mixins)	1,624	5.6%	Tightly coupled to App

2. Web Server: Reinventing the Wheel

The web server uses stdlib `http.server` (no async, no routing framework) with 1,656 lines of HTML templates as Python strings. This means:

- No template syntax highlighting or linting
- No asset pipeline (CSS/JS inline in Python)
- Manually parsing query strings and routing paths

- The `web_control_api.py` shells out to `subprocess` for everything

3. Session Manager: God Object Risk

`SessionManager` (834 lines) handles session CRUD, stats aggregation, budget management, heartbeat config, sleep state, archive state, oversight policy, and standing instructions — all in one class with one JSON file. It's becoming a dumping ground.

4. Daemon Coupling Via Shared Files

The monitor daemon writes `monitor_daemon_state.json`, the session manager writes `sessions.json`, the status history writes CSV files, the presence logger writes CSV files. The TUI reads all four. There's no schema versioning, no migration path, no validation at read time.

5. Fragmented "Pure Logic" Placement

Pure functions live in: `tui_logic.py`, `tui_helpers.py`, `tui_formatters.py`, `tui_render.py`, `monitor_daemon_core.py`, `supervisor_daemon_core.py`, `summary_columns.py`. The distinction between `tui_logic` vs `tui_helpers` vs `tui_formatters` is unclear — they all contain TUI-adjacent pure functions.

Improvement Ideas

High Impact, Lower Effort

A. Extract Web Templates to Real Files

Savings: ~1,600 lines deleted from Python, better tooling

Move HTML/CSS/JS from `web_templates.py` (Python string literals) to actual `.html` / `.css` / `.js` files loaded at runtime. This gives you syntax highlighting, linting, and the ability to iterate on the dashboard without touching Python.

```
src/overcode/web/
  templates/
    dashboard.html
    analytics.html
  static/
    style.css
    dashboard.js
    chartjs.min.js
```

B. Consolidate TUI Pure Logic

Reduce 4 files → 2

- Merge `tui_formatters.py` into `tui_helpers.py` (they already re-export each other)
- Rename `tui_logic.py` to something clearer like `tui_computations.py` or just keep it but add a clear docstring distinguishing it from helpers
- `tui_render.py` stays separate (it produces Rich `Text` objects, which is a distinct concern)

C. Make Summary Columns Data-Driven

Potential savings: ~400 lines

`summary_columns.py` (812 lines) defines each column with a render function. Many follow the same pattern: extract a field, format it, apply a style. A declarative column spec (field name, formatter, style rule) could replace most of these with a generic renderer.

D. Split CLI Into Submodules

Better organization, no line savings

```
src/overcode/cli/
  __init__.py          # app = typer.Typer()
  launch.py            # launch, list, attach, kill
  daemon.py            # monitor-daemon, supervisor-daemon
  config.py            # config, hooks, skills, perms
  data.py              # export, history, report
```

Each sub-module registers its commands on the shared `app`. This is standard Typer practice for large CLIs.

Medium Impact, Medium Effort

E. Replace stdlib HTTP with a Lightweight Framework

Better maintainability, enables async

Replace `http.server` + manual routing with something like Starlette or Litestar. You get:

- Declarative routing
- Proper request/response objects
- Template rendering (Jinja2)
- Static file serving
- WebSocket support (for real-time dashboard updates instead of polling)

This would eliminate most of `web_server.py` (642 lines of manual routing), simplify `web_control_api.py`, and make the web layer feel like a first-class subsystem rather than a bolt-on.

F. Split SessionManager by Concern

Reduce coupling, clearer ownership

`SessionManager` manages too many concerns. Split into:

- `SessionRegistry` — CRUD for session entries
- `SessionBudgetManager` — budget tracking and enforcement
- `SessionPolicyManager` — heartbeat, sleep, oversight, standing instructions

Each could operate on the same underlying JSON file but own a well-defined slice of the schema.

G. Schema Validation for State Files

Add Pydantic or dataclass-based validation when reading `monitor_daemon_state.json` and `sessions.json`. Currently, any field added/removed/renamed is a silent runtime error discovered by users. A schema version field + validation at read time would catch this during development.

Lower Impact, Higher Effort (Future)

H. Event-Driven Architecture

Replace file-polling with an event bus (even just Unix domain sockets or named pipes). The monitor daemon would publish events; the TUI, web server, and supervisor would subscribe. This eliminates the ~2s latency between status change and display, and removes the need for each consumer to independently poll and diff the state file.

I. Plugin Architecture for Status Detection

The status detection regex patterns are the most maintenance-heavy part of the codebase — every Claude Code UI change requires pattern updates. A plugin architecture where detection strategies are registered and prioritized would make this more extensible:

```
@detector(priority=10)
def hook_detector(session): ...

@detector(priority=5, fallback=True)
def polling_detector(session): ...
```

J. Unified State Store

Replace the four separate files (daemon state JSON, sessions JSON, status history CSV, presence CSV) with a single embedded database (SQLite). Benefits:

- Atomic multi-table updates
- Query capability (no more loading entire CSVs into memory)
- Schema migrations
- Single file to backup/export

Appendix: File Index

File	Lines	Subsystem
`cli.py`	2,449	CLI
`tui.py`	2,159	TUI
`web_templates.py`	1,656	Web
`monitor_daemon.py`	943	Monitor Daemon
`web_api.py`	844	Web
`session_manager.py`	834	Session
`supervisor_daemon.py`	821	Supervisor
`summary_columns.py`	812	TUI
`tui_logic.py`	685	TUI
`history_reader.py`	685	Data
`web_server.py`	642	Web
`tui_actions/session.py`	628	TUI
`tui_widgets/command_bar.py`	529	TUI
`web_control_api.py`	525	Web
`launcher.py`	524	Claude
`settings.py`	493	Config
`tui_actions/view.py`	475	TUI
`monitor_daemon_state.py`	459	Monitor Daemon
`status_detector.py`	457	Status
`presence_logger.py`	454	Data
`tui_widgets/session_summary.py`	450	TUI
`status_patterns.py`	442	Status
`tui_helpers.py`	436	TUI

File	Lines	Subsystem
`tui_render.py`	416	TUI
`monitor_daemon_core.py`	393	Monitor Daemon
`implementations.py`	348	Tmux
`tui_widgets/daemon_status_bar.py`	326	TUI
`time_context.py`	315	Claude
`follow_mode.py`	315	Claude
`supervisor_daemon_core.py`	314	Supervisor
`status_history.py`	308	Data
`tmux_manager.py`	303	Tmux
`config.py`	300	Config
`summarizer_component.py`	293	Claude
`tui_widgets/status_timeline.py`	289	TUI
`standing_instructions.py`	285	Claude
`sister_controller.py`	268	Sister
`testing/renderer.py`	268	Testing
`data_export.py`	257	Data
`hook_status_detector.py`	251	Status
`status_constants.py`	248	Status
`pid_utils.py`	246	Infra
`sister_poller.py`	238	Sister
`tui_formatters.py`	235	TUI
`testing/tmux_driver.py`	223	Testing
`exceptions.py`	219	Infra
`tui_actions/input.py`	217	TUI
`tui_actions/daemon.py`	201	TUI
`help_overlay.py`	197	TUI

File	Lines	Subsystem
`logging_config.py`	193	Infra
`protocols.py`	189	Tmux
`testing/tui_eye.py`	185	Testing
`claude_config.py`	186	Claude
`summarizer_client.py`	181	Claude
`bundled_skills.py`	175	Claude
`daemon_panel.py`	169	TUI
`summary_config_modal.py`	165	TUI
`mocks.py`	156	Tmux
`notifier.py`	145	Infra
`daemon_logging.py`	144	Infra
`summary_groups.py`	141	TUI
`fullscreen_preview.py`	130	TUI
`tmux_utils.py`	128	Tmux
`hook_handler.py`	123	Claude
`usage_monitor.py`	119	Infra
`dependency_check.py`	111	Infra
`web_server_runner.py`	106	Web
`tui_actions/navigation.py`	103	TUI
`daemon_utils.py`	93	Infra
`preview_pane.py`	101	TUI
`status_detector_factory.py`	83	Status
`interfaces.py`	49	Tmux
`web_chartjs.py`	32	Web