INDEPENDENT TECHNICAL REPORT · AUGUST 2026

Autonomous Personal Intelligence

A daily brief with an audience of exactly one: relevance computed against the reader's own goals, and an operating program governed like a codebase.

JOSEPH STEVENSON joseph-stevenson-portfolio.netlify.app

ABSTRACT

Sift is a personal intelligence pipeline I built to keep myself current on my own project portfolio: a daily, autonomously produced issue with an audience of exactly one. The design inverts the usual editorial stance. Relevance is not a judgment about the world; it is computed against the reader. Each morning a scheduled AI agent executes an operating program of roughly 1,400 lines, reads my goals, constraints, and live project state, scans a wide source pool against a 30-source floor, and ships a brief capped at 800 tool-verified words, in which an item appears only if it materially changes a goal or a planned action. The reader's side of the trade is arithmetic: a two-to-three minute read in place of a manual scan I estimate at two hours a day, with the relevance judgment precomputed against the reader's own goals. The system also ran a second experiment on itself: nearly every issue was scored against a rubric and an immutable vision contract by a CI grader, and sustained failures triggered a risk-routed pipeline proposing edits to the operating program, bounded by rollback tags, a diff cap, and a human-review queue. Sift ran daily for eighteen days, shipping on seventeen of them, including seven issues produced unattended while I was out of the country, and it survived one destructive incident that produced its safety contract. It is now dormant with its complete audit trail intact. This paper reports the architecture, the operating principles, and the gaps between specification and practice.

1 Introduction

An independent builder running several projects has a standing intelligence problem. Platform companies ship weekly. Any announcement might invalidate a plan, validate a bet, or open a consulting wedge; almost all change nothing. The standard instruments are built for audiences: newsletters, feeds, and AI summarizers compress the world for a median reader, so the real question, "does this change anything for me, today," stays manual, every morning.

Sift starts from the other end. It has an audience of exactly one, and it can read that reader's actual state: goals, finances, constraints, and the live status of every project. Relevance therefore does not have to be curated; it can be computed. An item ships only if the system can name the active project it touches and the concrete action it changes. Everything else is noise by

definition, however important to someone else. The operating program states the bar plainly: Sift exists to deliver items the reader "could not have found in the same time," with "the noise pre-filtered out."

The second experiment inside Sift concerns the program itself. Agent prompts are programs, and programs rot. Sift's operating program is treated as a governed codebase: version-controlled, protected by rollback tags, graded nightly on its output, and modified through a risk-routed editing pipeline with a human-review queue. This paper also reports what happened when that machinery met production for eighteen days.

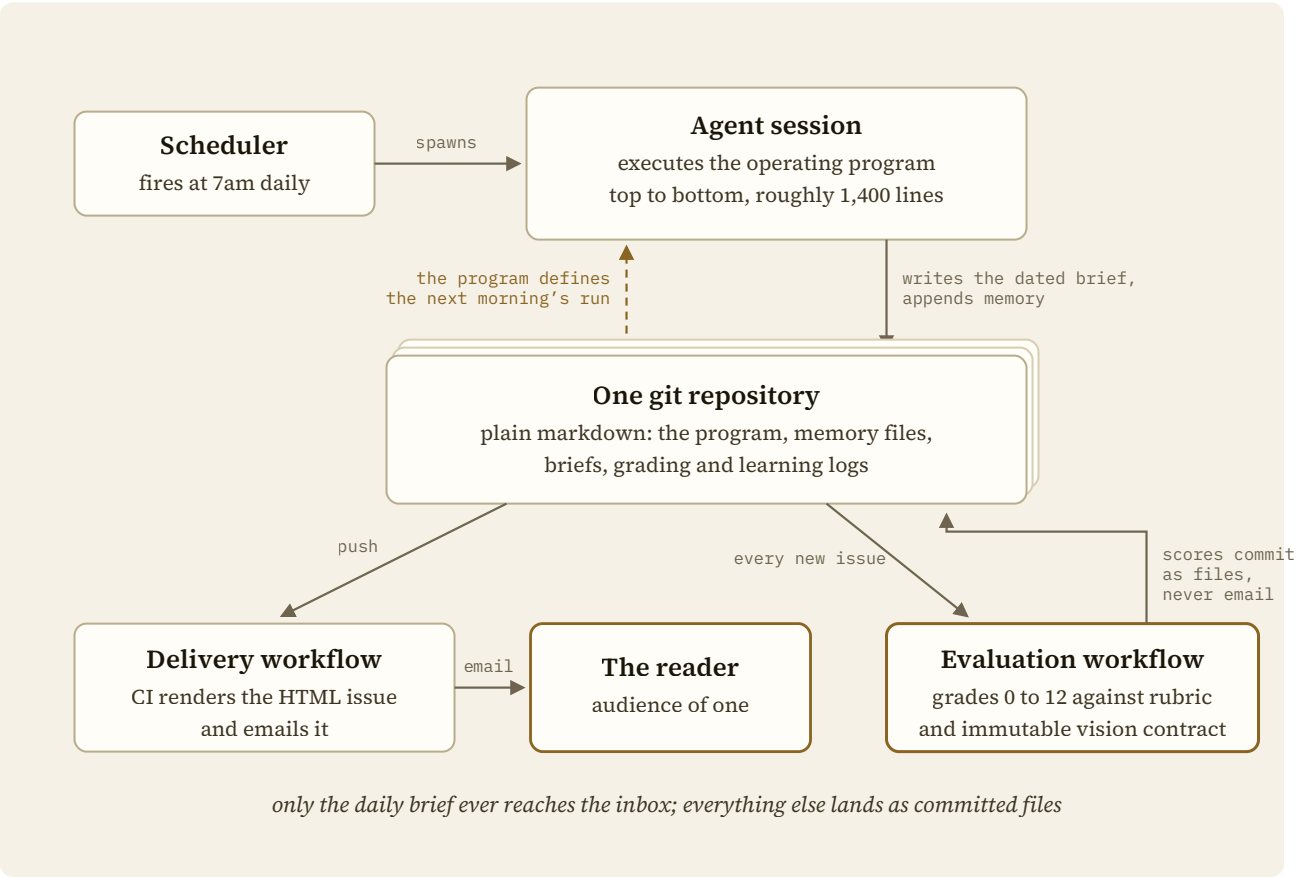
This is a design-and-field report, written from the artifact rather than from memory: archived briefs, grading and learning logs, an incident postmortem, queued proposals, and version-control history. The paper contributes: (1) a working design for reader-grounded relevance, in which an item ships only if it names the goal or planned action it changes, enforced by a verifier rather than an aspiration; (2) the governed-program pattern, an agent's operating prompt treated as production code, graded nightly on its output and edited through risk-routed lanes under rollback anchors; and (3) a complete field record, eighteen days of daily production with the unattended stretch, the destructive incident, and the specification gaps reported rather than smoothed. The direction also found external confirmation while the system ran: two major platforms shipped daily-brief products in this category mid-run, a convergence Section 5 records.

2 System Context

I am an independent AI systems builder, and Sift ran against my own portfolio: three active projects during the run, with older systems, including the multi-agent orchestration platform described in the companion report Agentic Orchestration Architecture, frozen in the entity database. The portfolio itself moved mid-run, one strategic commit renaming a consulting practice and archiving three older projects, which matters because the pipeline had to track a target that was itself changing.

The whole system lives in one git repository of plain markdown files. The daily producer was a scheduled AI agent session firing at 7am; delivery is a push-to-email chain, where the agent writes the dated brief into an archive directory, pushes, and a CI workflow renders it into an HTML newsletter and emails it to me. Tool integrations run over MCP, the open standard for connecting a model to external tools, for calendar writes and mail fallbacks.

FIG. 1



The system at a glance. One scheduled agent, one git repository of plain markdown, and two CI chains: delivery renders and emails the issue, evaluation grades it and commits scores as files. Only the daily brief ever emails the reader.

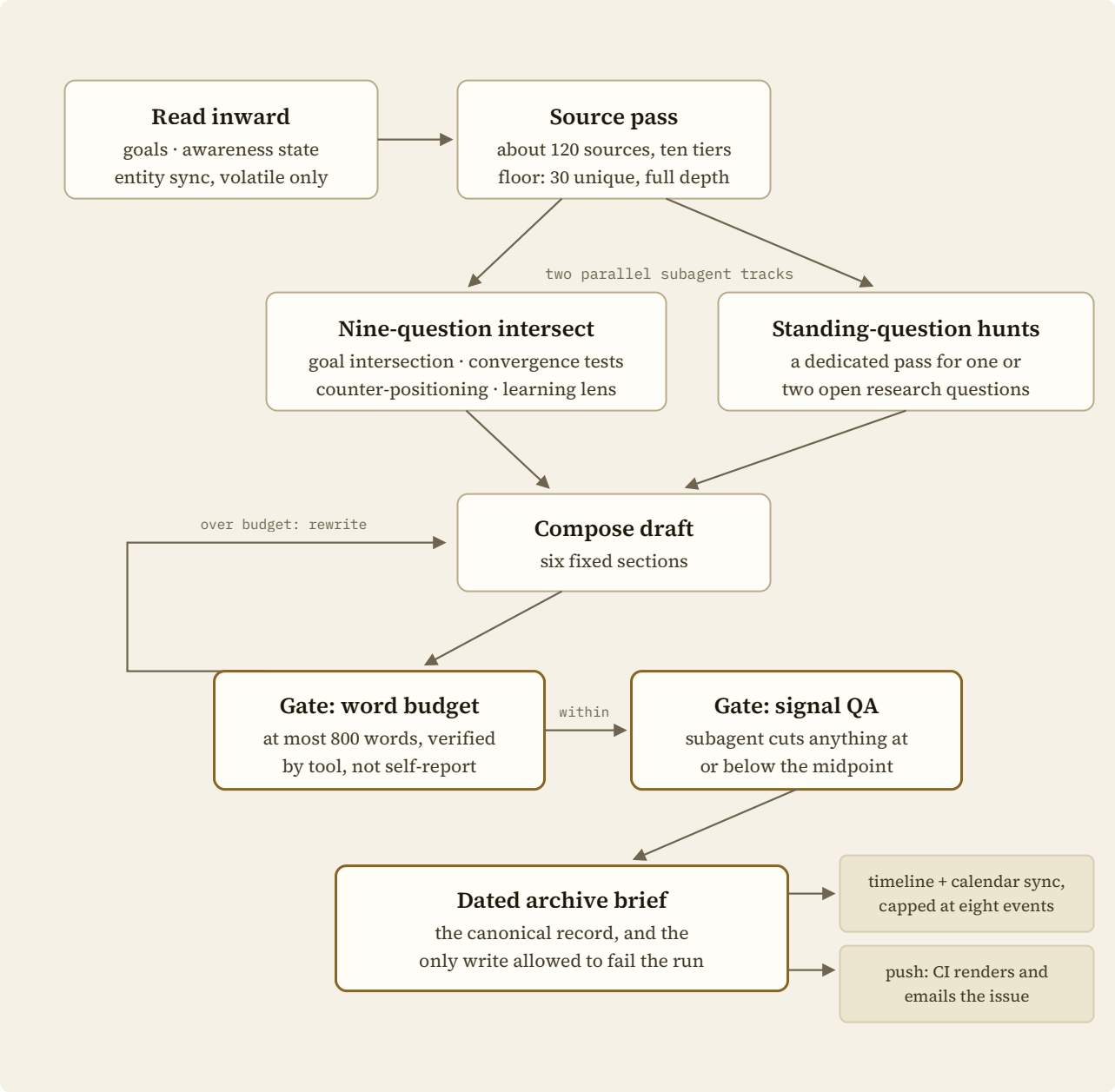
Three constraints shaped the design. One operator and no team: the pipeline had to run unattended. Auditability: I chose flat markdown files under version control instead of a database because the agent reads and writes markdown natively, every change is diffable, and rollback comes free. Portability: every pipeline script is dependency-free Python, so the cloud runner and my laptop run identical code with no environment setup. The cost envelope was measured in agent-minutes: a typical run spent between one and a half and three hours of agent time, with a hard four-hour ceiling.

3 Architecture

3.1 The daily run

The agent executes the operating program top to bottom every morning.

FIG. 2



One morning's run. The agent reads inward before it looks outward; two analysis tracks feed one draft; and two enforced gates stand between that draft and the archive write, the only write allowed to fail the run.

The run reads inward before it looks outward. Early phases pull yearly and monthly goals from the strategic layer, then load awareness state: rolling topic-cluster counters over the last week and month, a tracker of which past items I actually used, and calendar pressure that boosts an item's score when it touches a deadline within two weeks. Entity sync refreshes only the sections of each active project's file marked volatile, skipping archived entities; the sync was moved ahead of the reading pass after an audit caught the run reading stale project state.

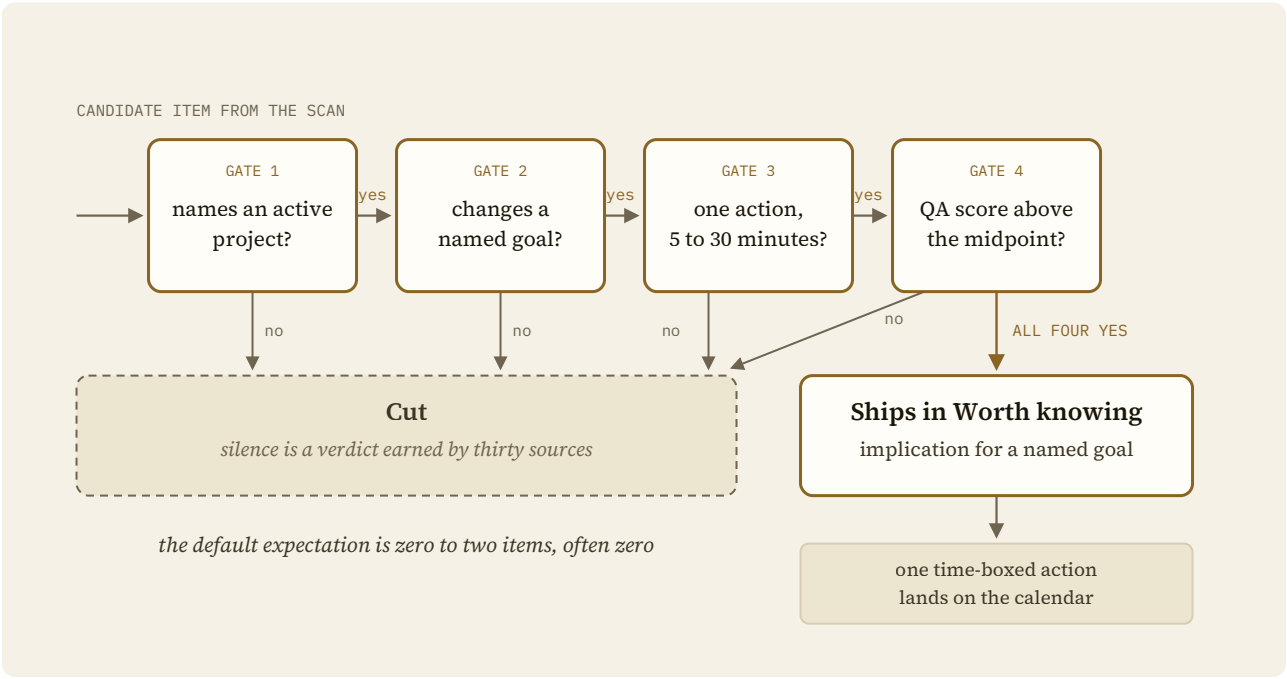
The outward scan enforces diversity: a pool of roughly 120 sources in ten tiers, every top-tier source daily, at least twenty rotating picks from the middle tiers, a floor of thirty unique sources per run, no back-to-back repeats unless a source scored highly, plus pre-announcement feeds (careers pages, patents, conference talks) and surveillance of named prospect businesses. Two analysis tracks then run as parallel subagents. The first is a nine-question intersect per item: goal and ticket intersection, convergent-invention tests in exact, adjacent, and inverse forms, competitive overhang, constraint triggers, a counter-positioning question that drafts a verbatim pitch line, and a learning lens over a curated library of twenty operating-framework books and six case studies, which ships a lesson only if it clears four of five binary gates. The second is a dedicated hunt for one or two standing research questions.

Composition assembles a draft, and quality control runs in two stages: a mechanical, tool-enforced gate that blocks the run and forces a rewrite when the draft breaks the word budget (Principle 9), then a signal-versus-noise QA subagent that cuts anything scoring at or below the midpoint of its scale. Its prompt quotes me directly: "if it is shit, I don't want to see it." The surviving draft is written to the dated archive file, the canonical record and the only write allowed to fail the run. Final phases materialize consequences: a machine-generated timeline and a calendar push, idempotent via tagged event ids, conflict-aware, and hard-capped at eight new events per run.

3.2 The brief

The product is at most 800 user-facing words, a two-to-three minute read, in six fixed sections: where I am headed, goal status against the month and the year, this week's plan, today, worth learning, and worth knowing. Today carries one highest-leverage move with a numbered gameplan, selected by internal probability-times-upside scoring that never appears in the output. Worth learning applies exactly one book or case-study lesson to the current week. Worth knowing holds zero to two news items, each with a one-line implication for a named goal, at most two citations, and exactly one time-boxed action of five to thirty minutes. The CI renderer softens jargon, guards against placeholder leaks, and silently drops any section heading it does not recognize.

FIG. 3

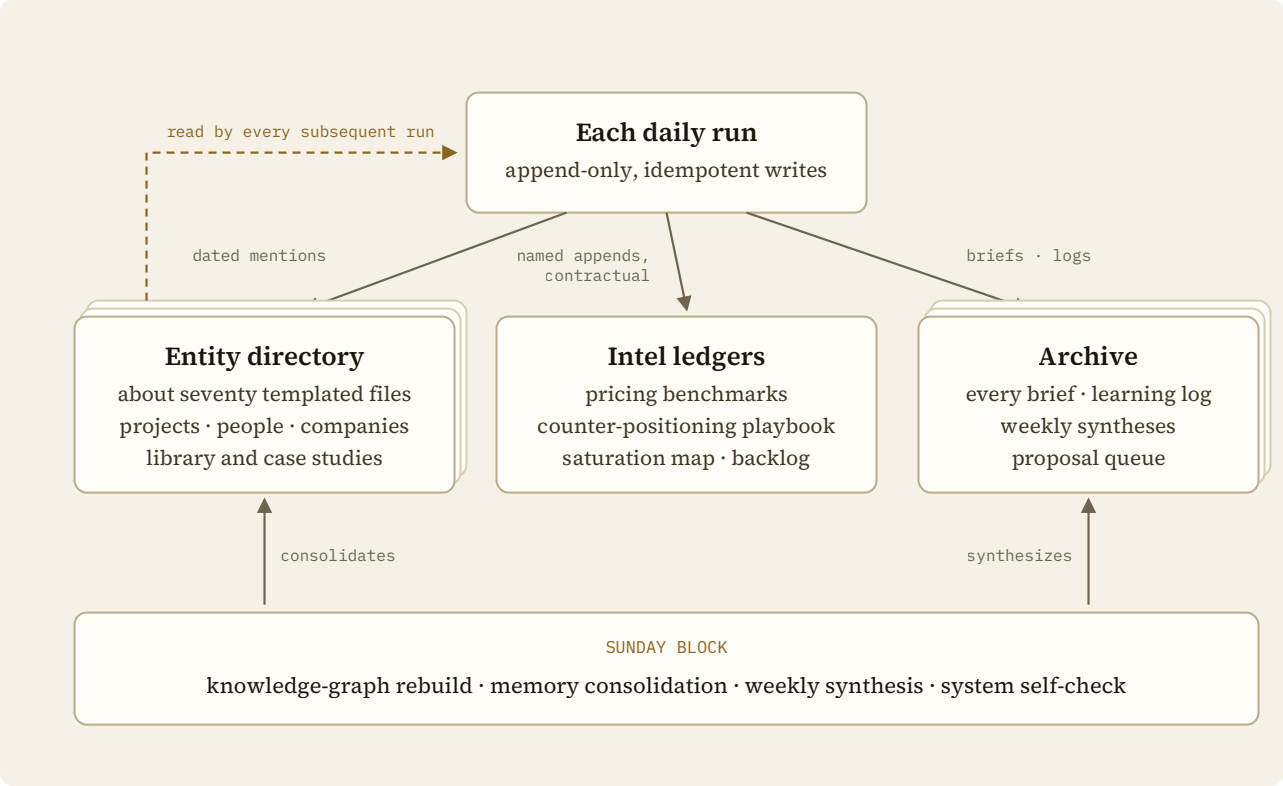


One item's gauntlet. A candidate item ships only if it survives four gates in a row; the default outcome is the cut, and a shipped item lands on the calendar with a time-boxed action.

3.3 The memory layer

Durable state lives in three structures. An entity directory holds about seventy markdown files: projects, companies, people, concepts, the twenty-book library, and the case studies, each on a strict template with an append-only dated mentions log, idempotent on same-date re-runs. An intel directory holds compounding ledgers for the consulting practice: pricing benchmarks, a counter-positioning playbook with verbatim pitch lines, a saturation map, and an infrastructure backlog; every brief must reference an intel file by name, and named appends are contractual before a brief counts as shipped. The archive is the append-only record: every brief, the learning log, weekly syntheses, and the self-edit proposal queue. A Sunday block adds a knowledge-graph rebuild, memory consolidation, and a weekly synthesis ending with a system check of Sift itself.

FIG. 4

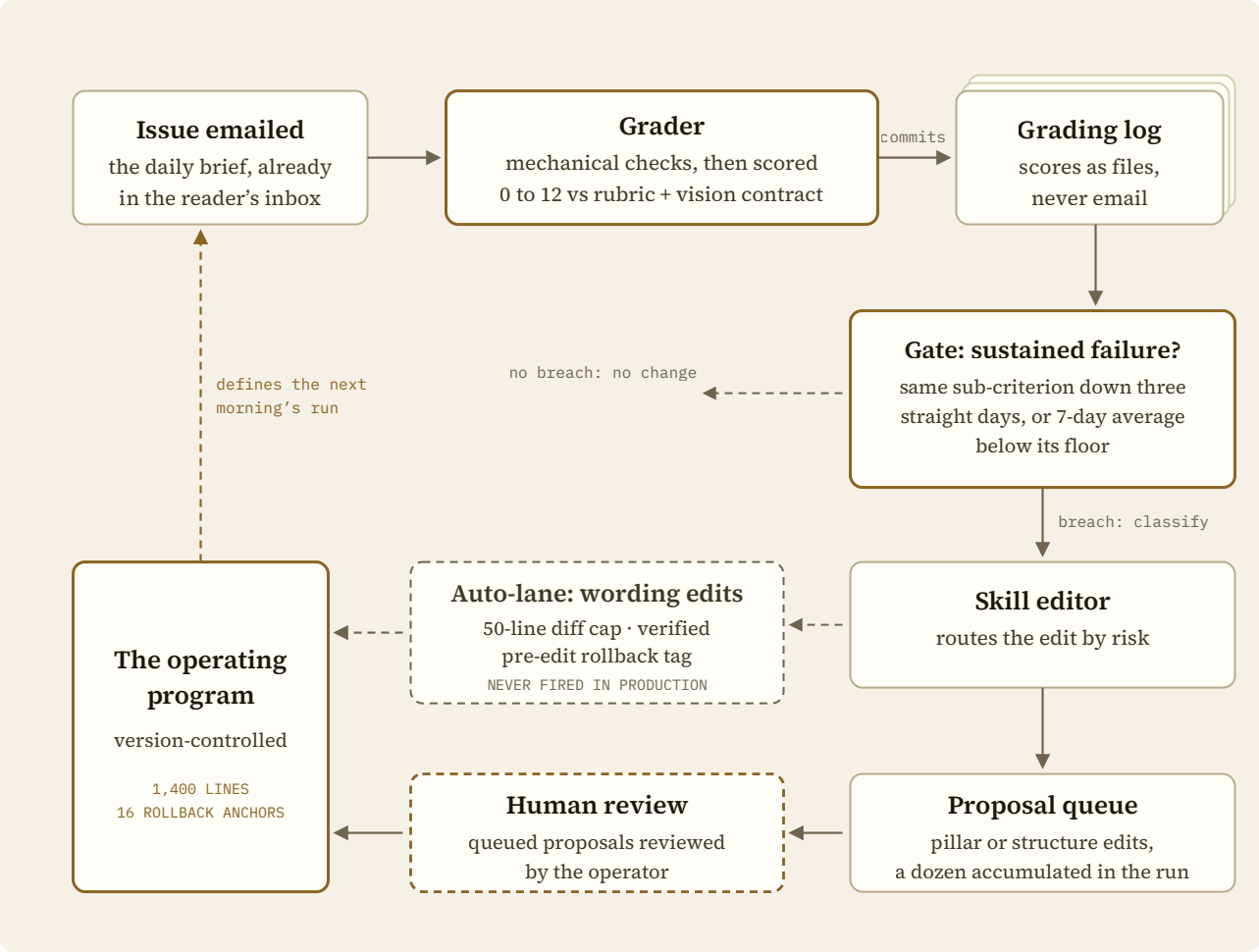


The memory layer. Every run appends to three durable stores under append-only, idempotent discipline; a Sunday block consolidates them; and every subsequent run reads the accumulated state back.

3.4 The self-correction chain

Delivery and evaluation are chained CI workflows.

FIG. 5



The governance loop. Graded output drives edits to the operating program through risk-routed lanes: wording fixes may auto-apply under a diff cap and rollback tag, while structural edits queue for human review. In production, every trigger routed to the human queue.

One workflow emails each new brief; a second re-renders it, runs mechanical pass-or-fail checks (the lead must name an active project and no archived one; market tagging, jargon, and intel-file references are verified), scores it 0 to 12 against a written rubric and an immutable vision contract, and commits the grading log. If the same sub-criterion fails three or more consecutive days, or the seven-day rolling average falls below its floor, the skill editor proposes edits to the operating program itself. Wording-level fixes are auto-applicable behind a 50-line diff cap and a verified pre-edit rollback tag; anything touching scoring pillars or brief structure is committed quietly to a proposal queue for my review. A dozen proposals accumulated over the run. By deliberate preference, only the daily brief ever emails me: the grader has no notification permissions, and everything it produces lands as committed files.

3.5 The convergent-invention protocol

Sift's signature detector formalizes a pattern that matters to a solo builder: I ship a feature, and a platform company independently ships the same thing days later. The protocol requires both ship dates, the platform's within 60 days after mine, and a substantive pattern match documented in one or two sentences. Research previews are excluded as anchors, and pairs whose timing is uncertain by more than a week are flagged as timing-ambiguous and require my confirmation before they count. The anchor instances: a health-platform feature I shipped was matched by the wearable vendor's equivalent sixteen days later, and the orchestration system described in the companion report was matched in six of seven patterns by a platform announcement one day after its public ship. The protocol also removes entries: one platform pair was struck from the ledger when investigation showed the platform had shipped seventeen days before me, the learning log recording the reason as "intellectual honesty first."

4 Core Principles

Thirteen principles held up across the run. Each carries the evidence that made it a rule.

1. Relevance is computed against one reader. Every shipped item must name an active project and a project-specific implication; the program states that Sift is "a tailored newsletter, not a generic AI-landscape feed." This is enforced, not aspirational: the CI verifier fails any brief whose lead names no active project or leads with an archived one.

2. Breadth of scan, narrowness of ship. The wide source pass runs at full depth daily precisely so the brief can credibly say nothing happened. The default expectation is zero to two news items, often zero. The morning after the largest AI-industry valuation story of the season broke, the brief shipped exactly two items. Padding is a defect; silence is a verdict that took thirty sources to earn.

3. Every event gets a verdict and a clock. No item ships as raw news. Each carries a one-line implication for a named goal, at most two citations, and exactly one five-to-thirty-minute action; one mid-run issue named the exact demo-script sentence to rewrite. Actions then flow into the machine-generated timeline and onto the real calendar under idempotency tags.

4. The operating program is a governed artifact. The program is not documentation; it is the system, treated like production code: file-of-record protection, edits verified by re-rendering past briefs, and a pre-edit rollback tag before substantive changes. Sixteen rollback anchors accumulated in eighteen days.

5. Graded output closes the loop. Program evolution is triggered by measured output quality, not taste: scores ranged from 5 of 12 to 12 of 12 across the run, and edit proposals fire only on defined threshold breaches. The final trigger of the run was a seven-day rolling average falling a tenth of a point below its floor.

6. Self-editing is routed by risk. Wording drift may be fixed autonomously under a capped diff; anything touching pillar enforcement or brief structure goes to the human queue, and the last queued proposal of the run says in its own text, "DO NOT auto-apply this." A vision file that code reads and never writes bounds the loop. In production every trigger classified as high-risk, so the autonomous lane never fired: in hindsight, a healthy outcome.

7. Guardrails must be verified, not described. Mid-run, a build agent testing the self-editor against the live program destroyed roughly forty lines of uncommitted work with a cleanup checkout, then deleted the rollback tag that was supposed to protect them. The resulting contract is blunt: "Documented guardrails without verification are not guardrails." A spec line claiming a tag gets created counts for nothing until a test asserts it exists, and audit trails may never be deleted as cleanup.

8. Test on copy, never on live. Version-control tags point at commits and cannot protect uncommitted working-tree state, so any script that writes tracked files is tested only against a disposable copy, and all write-capable tools refuse live targets without an explicit consent flag. The destroyed edits came back only because the session logs allowed replay: recovery by luck, converted into a rule so it would never again require luck.

9. Tool output over self-report. The composer once reported 800 words when the tool counted 952, and a 2,073-word issue shipped past a soft rule. Composer self-checking is now banned outright: word counts, banned-phrasing scans, and file-existence preconditions must be verbatim tool output pasted into the learning log in a literal format. Evidence became structurally distinguishable from assertion.

10. Honest negatives are the product. The struck convergence pair, the timing-ambiguous gate, logged null results from standing-question hunts, and a data point retained despite weakening my own product's claim all follow one logged rule: the credential is more durable with honest guardrails than without them.

11. Anti-examples are legislation. Nearly every binding rule embeds the failure that created it. Recommending outreach to a developer-relations function that turned out not to exist produced the validate-before-asserting rule; a false "demo script is complete" claim produced a tool-enforced existence check; a public-facing move recommended during a deliberately low-key phase produced a sequencing rule. The program reads as accumulated case law.

12. Value accrues in compounding files, not transient briefs. A brief item counts only when it lands in an append-schema asset: intel ledgers, entity mentions logs, and a downstream-reference tracker that converts items produced into items used. The weekly synthesis reports items that compounded as its outcome metric.

13. Fail loudly, never fail the run. Every phase has an explicit failure budget, twenty-one enumerated modes; only the canonical archive write may abort a run; everything else degrades with a visible setup note. The operating contract sets the norm for agents and humans alike: "Silence on destruction is not acceptable behavior here."

5 Eighteen Days of Evolution

The versioned record spans eighteen days of production, roughly a hundred commits, and it divides into a bootstrap, a hardening, and a test.

The bootstrap day swept a month of baseline across some fifty sources, enriched the seed entities, and investigated ten standing questions. It also produced the canonical failure: the first brief recommended reaching out to a developer-relations function without checking that the function exists. That mistake became Principle 11's first entry, and the validate-before-asserting rule it created was enforced by tooling within days. The first subtraction came almost immediately after: an early feature that modulated the brief's length against biometric recovery data was deleted outright rather than patched, the first evidence that the program would be governed by removal as well as addition.

The hardening arrived in the middle of the run, and one day of it was pivotal: the destructive-git incident of Principles 7 and 8 happened while building the self-editor, and the same day shipped the self-grading loop, the timeline and calendar materialization, the tightened word budget, and the curated library wired into the intersect. The days that followed hardened the gates further: a mechanical word-count gate after an issue shipped at two and a half times budget, a goal-state freshness check after a brief recycled framing that the goals file had superseded a day earlier, and tool-enforced banned phrasings after retired labels shipped past a ban that was nominally live.

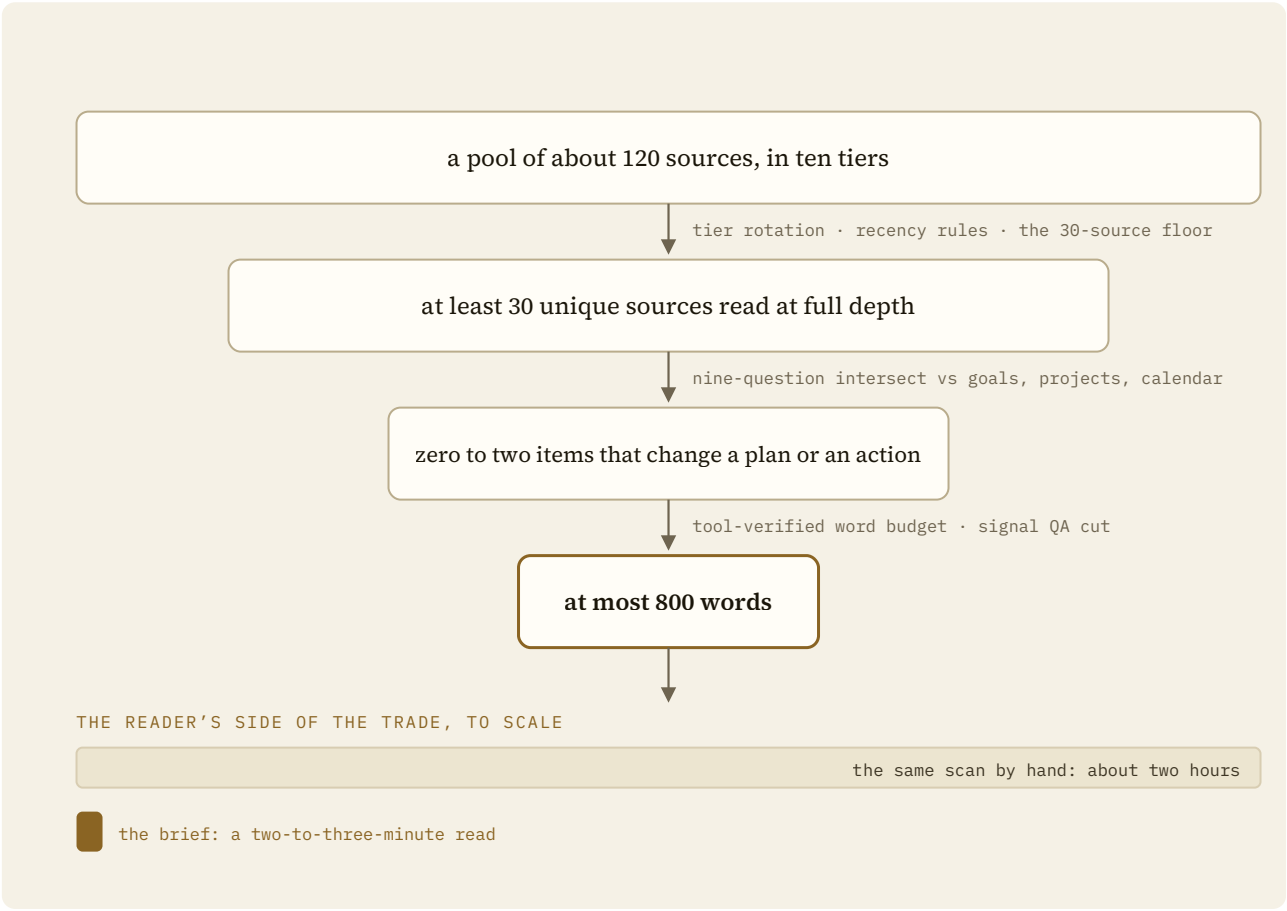
Then the system passed its real test. I left the country for ten days; Sift shipped seven issues unattended, the mid-trip leads correctly reading that the reader was away and adjusting the day's one move accordingly. CI graded the issues and queued edit proposals throughout, with one honest gap: one unattended issue was never scored. The scan also caught the market converging on Sift's own category while it ran: within one week, two major platforms shipped daily-brief products adjacent to what this system already did for one reader. In the run's final days I rewrote the vision contract from news-awareness to goal-execution ("keep me in the loop, allow me to learn from similar case studies or frameworks, while also pushing me closer to my

goals"), re-aimed the grader, and reshaped the brief. The first issue under the stricter rubric scored 11 of 12. It was also the last: the record ends there, with no shutdown note, and I report that plainly rather than reconstruct a reason.

6 Instrumentation and Evaluation

The instrument's arithmetic frames everything else. A run reads at least thirty unique sources at full depth, spends between ninety minutes and three hours of machine time, and delivers at most 800 words: a two-to-three-minute read. Performed by hand, the same scan is on the order of two hours of reading a day, an estimate I state as one; the brief replaced it with minutes, with the relevance judgment already computed against my own goals rather than left as homework. Across the run, seventeen issues covered more than five hundred source-reads and asked of me a total reading time of under an hour, and seven of the seventeen (41 percent) were produced with zero human minutes while I was out of the country. The compression only counts if what survives it is right, which is what the measurement layers below exist to check.

FIG. 6



The compression arithmetic. A pool of about 120 sources narrows, through the intersect and two enforced gates, to a brief of at most 800 words; the reader's side of the trade is a two-to-three-minute read in place of an estimated two-hour manual scan.

Sift measured itself at three layers.

At the issue layer, a verifier ran mechanical pass-or-fail checks and a deterministic grader scored four pillars from 0 to 3 each: after the late-run re-aim, goal advancement, action concreteness, learning applied, and clarity with honest signal. The grader always exits cleanly so grading can never block delivery, and its heuristic counts only action verbs in imperative position, so past-tense news narration cannot masquerade as a plan. An LLM-judge path exists in the code and never ran; every score came from the deterministic heuristic, and the grading log says so on every line. Coverage was good but not perfect: a small number of briefs were never scored, and the gaps are visible in the log rather than backfilled.

At the program layer, threshold breaches drove the skill editor, which queued a dozen high-risk proposals, pushed rollback tags to the remote so anchors survive any single machine, and produced weekly scoreboards as files, never email.

At the value layer, instrumentation was honest but thinner. Topic-cluster counters tracked saturation, demoting any cluster that had already produced three items in a week, and a reference tracker was built to measure which items I later used. That measure matters most and proved hardest to automate: the cloud runner could not see into my local working directories, so evidence of compounding was collected manually and sparsely.

7 Limitations and Open Problems

- Production self-modification never happened. The loop ran end to end daily, but every trigger classified as high-risk, the auto-apply lane never fired, and none of the queued proposals were merged. All substantive program evolution went through me in interactive sessions. Whether to harden the autonomous lane or accept proposal-only as the right equilibrium remains open.
- The LLM grading path was built and never exercised; every score came from the deterministic heuristic. It worked for trend detection but measures form more than judgment.
- The evaluation is a sample of one in every direction: one reader, seventeen briefing days, no control condition, and only partial evidence on whether the briefs' recommended actions were taken.
- Spec-versus-reality gaps are visible in the artifact: a mandated canvas directory sits empty, one contract-named intel ledger was never instantiated, the first monthly review never occurred because the system stopped before its date, and one phase block is an explicit placeholder. I catalog these rather than hide them because they are the honest shape of an eighteen-day system.
- Nothing in the repository documents why daily runs stopped. The record ends at the vision re-aim, and I report that plainly.
- Email delivery is evidenced by workflow definitions and configuration, not receipts: the explicitly logged delivery events are early failures and one fallback draft, and no receipt evidence exists for the daily CI sends.
- The scheduler itself lives outside the repository: the daily trigger is documented but not versioned with the artifact it triggers.

8 Conclusion

Sift's two claims survived contact with daily production. First, when the audience is one person whose goals, constraints, and project state are machine-readable, relevance stops being an editorial act and becomes a computation, and the correct output of a wide daily scan is usually

silence with receipts. Second, an agent's operating program can be governed like a codebase: graded on its output, modified under rollback anchors and risk routing, and hardened by rules that each carry the failure that created them. The costs were real: the safety contract was paid for with an actual incident, and the autonomous half of the self-editing loop turned out to be the part I still would not delegate. The system is dormant, but dormant with its complete audit trail intact, and for an eighteen-day experiment run by one person, the trail is the result.

A Note on the Record

This paper is written from the artifact rather than from memory: the operating program and its rubric, the immutable vision contract, twenty-two archived briefs, the grading and learning logs, the queued self-edit proposals, the incident postmortem written into the operating contract, and the version-control history with its sixteen rollback anchors. Where the text compresses dated events, the underlying logs carry the dates and diffs. The companion reports, Agentic Orchestration Architecture on the multi-agent orchestration platform and The Approval Desk on the human-gated outbound sales pipeline, draw on the same record-keeping discipline; the three systems share one design culture, and this paper is its field diary.