

CANDIDATE BACKGROUND	SYSTEM STATUS	TARGET USER	CORE MODEL
Engineer (Temenos SWE) → MBA (IIT Kharagpur) → Enterprise BD, New Logos (Hexaware)	In Production (Hourly Runs Since Sep 2026)	Solo Operator (Userbase: 1)	DeepSeek-V41-Flash (API)

The Self-Maintaining Knowledge Base

Designing an autonomous ingestion, verification, and evaluation pipeline for a solo AI product builder.

SYSTEM CONTEXT & CORE OUTCOME

Building multiple complex AI projects concurrently (EaseMySemester, Clarity, Semantic Search, Portfolio) creates persistent documentation debt. Manual documentation fails because shipping features always takes priority over updating notes. I built an unattended pipeline that runs hourly via Windows Task Scheduler. It harvests commits, PRs, and design notes across project repos, sanitizes private data through a two-pass redactor, synthesizes clean wiki pages, evaluates claims with a toolless checker agent, deterministically lints against retired claims, and exports verified facts to my public portfolio. It eliminates documentation toil while maintaining ground truth auditability. What follows is one question changing shape four times: what the pipeline is allowed to read, whether what it writes is true, what happens when a run dies halfway through, and what the whole thing costs. Each answer is a thing I stopped trusting a model to do.

204 Pages Written and Checked Counted at the 22 Sep export. A live figure that grows on every run, with no manual writing.	71 of 204 Graded Publishable Only Tier A pages reach the public site. Counted at the same export; the other 133 stay in the local vault.	~\$0.03 Metered Cost per Batch One write and one checker call over five pages, metered across 112 calls in a log that starts 20 September. A current reading, not a settled average.	~21,000 Thinking Tokens, Uncapped Checker Spent to make three sentence edits, before effort was budgeted per step. One observation, not a benchmark.
--	--	--	--

Problem Space: Why Solo Documentation Drifts

Before any of that, the failure worth designing against. Documentation does not rot for one reason, it rots for two, and they pull in opposite directions: the first is about time passing, the second is about what happens the moment you point a model at the problem.

1. The Reality Gap (Temporal Drift) In fast shipping cycles, code changes daily: pricing tiers disappear, features get refactored, and model choices evolve. Traditional documentation relies on memory to update old notes. When stakeholders read stale docs, contradictions destroy trust. Root Cause: Writing documentation is a secondary task that competes directly with building product features.	2. Uncontrolled AI Summarization Connecting an LLM directly to personal workspaces risks leaking client PII, internal financials, or unreleased partner notes. Standard generative models also round numbers and invent unearned claims ("drastic speedup", "seamless UX"). Root Cause: LLMs naturally optimize for persuasive prose rather than ground-truth factual constraints.
--	--

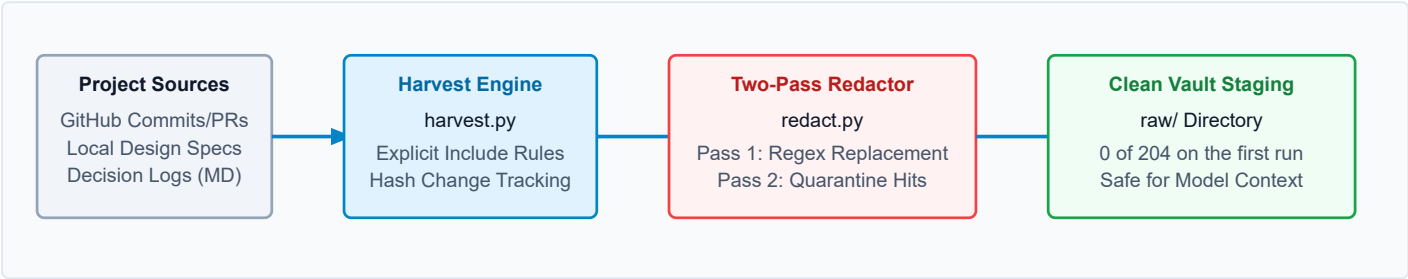
Product Principles for Solo Automation

Those two failures set three rules, and every design decision in the rest of this document is one of them being paid for.

- Zero human-in-the-loop for routine sync:** No step may prompt for confirmation or block execution. If a run fails, it parks state and exits cleanly.
- Separation of generation and grading:** The model that drafts a page cannot be the model that verifies it. Grading requires strict toolless constraint checking.
- Deterministic enforcement over prompt pleading:** Never ask an LLM to remember styling, word caps, or banned buzzwords. Enforce them in deterministic code.

System Architecture: The Ingestion Perimeter

The first rule, no human in the loop, sounds like a convenience. It is actually the expensive one, because it means nobody is standing between a source folder and the model to catch what should not have been sent. The pipeline reads across the project repositories and local planning folders, and those folders hold commercial artifacts: clinic documents, enterprise work notes. So the ingestion layer has to be the thing that would otherwise be a person saying stop.



Two-Pass Redaction Layer

One pass would be a filter. Two passes, the second one deliberately ignorant of the first, is the same maker and checker split that governs the writing, applied to privacy instead of truth.

Pass 1: Structural Replacement

Transforms known sensitive shapes into static tokens: private keys, GitHub tokens, database connection URLs, phone numbers, and clinic entity names are converted to markers like [secret], [db-ur1], and [phone].

Rule: The writer prompt is strictly forbidden from copying any bracketed placeholder into generated pages. The deterministic linter rejects any page containing a placeholder token.

Pass 2: Residual Quarantine

A completely separate scanning pass with broader heuristics inspects the output of Pass 1. If an uncaught email address, unexpected monetary figure, or patient identifier is detected, the item is not written to staging.

Action: harvest.py:68 quarantines that specific item into rep['quarantined'] and carries on processing remaining clean files. No sensitive payload reaches the model.

Data Governance: The Three Information Tiers

Redaction removes what must never be stored. It cannot decide what is safe to publish, because that is a judgment about audience rather than a pattern in the text: a defensible fact can be perfectly safe in an interview and wrong on a public site. So every page in the vault carries a mandatory tier, and the tier travels with the page for the rest of its life.

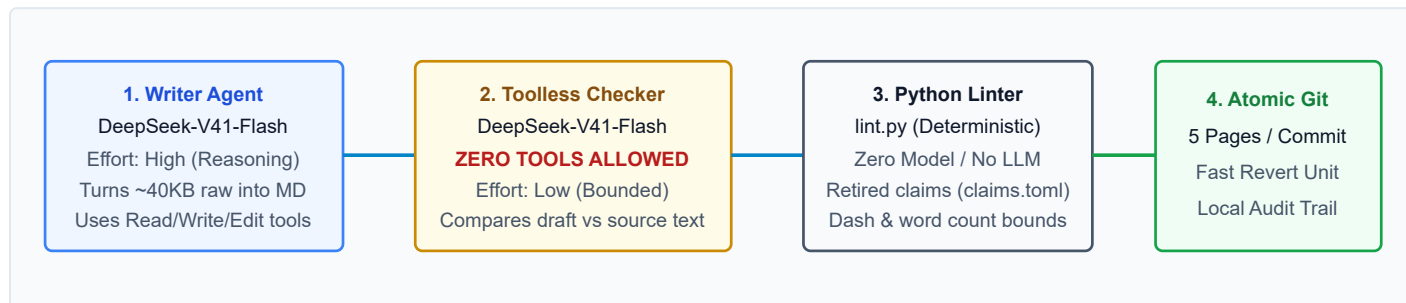
TIER	DEFINITION	CONTENT SCOPE	PUBLIC AVAILABILITY
TIER A	Publishable with Caveat	Shipped capabilities, verified metrics, backtest results, and architectural tradeoffs. Must always travel with its specific caveat.	Exported to portfolio site via export_site.py.
TIER B	Interview Only	Gaps, open bugs, unshipped plans and their phase status, and evidence that rests on a single conversation.	Vault only. Never published online.
TIER C	Strictly Confidential	Patient data, clinic financials, infrastructure endpoints, named clients and employers, job search applications, and compensation.	Never leaves the vault. The vault is local, so these pages are committed but never exported.

SOURCE HIERARCHY CONTRACT

When sources conflict, the harvester resolves disputes through a strict hierarchy: **1. Working Code & Git Commits > 2. Decision Logs (DECISIONS.md) > 3. Specs & READMEs > 4. Legacy Narrative Notes.** Newer sources defeat older sources at the same tier.

Evaluation Architecture: The Maker / Checker Split

Tiering settles what a page is allowed to say to whom. It says nothing about whether the page is true. In earlier prototypes a single model drafted a page and then verified its own text. It consistently justified its own hallucinations, because the cheapest way for a model to pass its own check is to decide that the check passed. **Splitting maker from checker is the fix, but the split only holds if the checker is also denied the means to go looking for agreement.**



Why the Checker Has Zero Tools

Denying the means to go looking for agreement is a specific, and slightly counterintuitive, engineering decision: the evaluator is the one agent in the system that gets less capability, not more.

Preventing Confirmation Bias

When an evaluator model has search tools (grep, glob, file reading), it queries the workspace until it finds text resembling its draft claim. This creates artificial verification.

Design Fix: The Checker receives only the draft page and the exact cited source snippet in the prompt. It has zero tools to navigate elsewhere, forcing a strict test of factual entailment.

Deterministic Linting as Grounding

LLMs are unreliable at enforcing negative and structural rules (such as word limits or forbidden characters).

Design Fix: `lint.py` runs without an LLM. It validates frontmatter schema, verifies cited source paths exist, asserts page length (150 to 500 words; cap 900), catches banned jargon, and rejects retired claims.

Token Economics: What the Checker Actually Costs

Taking tools away from the checker bounded where it could look. It did not bound how long it would think about what it found. In early testing an uncapped checker spent **~21,000 thinking tokens** to make three sentence edits, which is the same confirmation seeking behaviour arriving through a different door. Setting effort per step, rather than a token ceiling, keeps the checker on the comparison without cutting its output off mid thought.

Effort Budgeting Profile

- Writer:** effort = "high" (synthesizes multiple sources, resolves nuances).
- Checker:** effort = "low" (direct comparison between draft and source).
- Repair:** effort = "off" (applies literal linter issue fixes).
- The ceiling is soft:** effort steers, it does not cap. The largest single spend in the log is still a checker call at 59,444 thinking tokens.

Measured Costs from model-calls.jsonl

- Writer:** \$0.021 mean per call, \$0.024 median, over 23 calls.
- Checker:** \$0.011 mean per call over 70 calls. 29% of its input came from cache, computed from the log rather than reported by it.
- Repair:** \$0.002 mean per call, zero thinking tokens.
- Typical batch:** ~\$0.03, being one write plus one checker call over five pages.
- Observed daily spend:** \$0.83 on 21 Sep and \$0.48 on 22 Sep, the two full days in the log.

The log only starts on 20 September and contains no Sunday audit run, so what a full reread costs is still unmeasured. That number, rather than the daily total, is the one that decides whether a weekly audit stays affordable as the vault grows, and it is the next thing worth instrumenting.

Operational Resilience: The Resumable State Machine

Everything so far assumes the run finishes. A batch that dies halfway is worse than one that never started: re-running it spends the tokens twice and can write a second, divergent version of a page that was already correct.

Parked State Recovery (.state/parked/)

Pages are committed in batches of five as they pass linting. If a call comes back at the usage limit:

- Uncommitted pages move into .state/parked/ and the run aborts with an exit code.
- The next hourly run restores the parked files and resumes directly from the interrupted verification step.
- Finished commits remain intact.

Public Site Export (export_site.py)

The pipeline exports public metrics to my portfolio site.

- Pinning:** facts.toml ties each public metric to one wiki table row.
- Sub-tier Stripping:** a Tier A page may hold Tier B rows; the exporter strips the rows rather than blocking the page.
- Alarm Principle:** If a pinned wiki row changes shape, the export build fails, preventing silent metric divergence.

What I Deliberately Did Not Build

Parking and resuming is what the system does when it fails. These three are what it never attempts, each the same trade: less capability for a system that still runs in six months.

- No review gate:** nothing in a run waits on a person, and there is no approval step or PR to merge. Safety comes from the toolless checker, deterministic linting, and git history as an undo button. A process needing daily approval would be abandoned.
- No hosted vault:** The vault is a local git repository with no remote, so nothing can push it anywhere. Only curated Tier A pages leave via export to /wiki on the portfolio site.
- No automated public copy generation:** the pipeline produces grounded facts, caveats and tiers. Anything read publicly, including this document, I write myself.

Product Retrospective: Handling Contradictions and Temporal Drift

Refusing a review gate leaves one failure unassigned. Contradiction between live sources is the easy case, and contradictions.md tracks it. The hard case is **temporal drift, where no source changes and the world does**: every page still matches its evidence, and the evidence is out of date.

The claims.toml Registry

Maintains retired claims as regex patterns (e.g. past paying student counts for EaseMySemester v1). If an old narrative page repeats a retired claim in present tense, linting halts export until repaired.

Weekly Rotating Audit

Every Sunday, the checker rereads an entire project group in full against its sources, rotating weekly across EaseMySemester, Clarity, Semantic Search, Portfolio, Wiki, and Narrative to prevent slow drift.

System Review: Tradeoffs and Open Questions

Each of the decisions above bought something and cost something. The ledger, with the part I have not solved in the right hand column.

AREA	ENGINEERING / PRODUCT TRADEOFF	OPEN LIMITATION
Toolless Evaluation	Checker receives only page + source in prompt. Prevents confirmation bias searches.	Checker proves a page matches its source; it cannot prove whether the underlying source is still true.
Reconciliation	Contradictions between live sources are logged to contradictions.md as the checker meets them.	When two repos disagree on dates, human intervention in the processing order (wiki.toml [order]) is still required.

HARVEST AND VERIFY

CITE AND TAILOR

WHAT THE VAULT IS FOR

The vault exists so that every number on my public site has a row behind it, carrying its caveat and the file it came from. A second scheduled loop may read that row when it drafts an application. That is the whole relationship.