PERSONAL PROJECT · CASE STUDY

SEEK Agent

An autonomous job-application agent that decides what to apply for, which résumé to send, and how to answer an employer's questions — built around one hard rule: it is allowed to cost you the job, but it is never allowed to lie to get it.

`Python` `Playwright` `Claude API (Sonnet)` `JSON local memory` `202 automated tests` `3,716 LOC · 38 functions` `Jira · Confluence`

01 The problem, and the idea

Applying for jobs at volume is mostly repetition. The same forms, the same questions — *"How many years of experience do you have with X?"* — across hundreds of listings, most of which were never relevant. The obvious automation is a bot that fires off applications as fast as it can. That build is wrong twice over: it floods employers with noise, and the moment a form asks a question the applicant can't answer well, a volume-optimised bot has every incentive to round up.

This is the opposite. The agent applies **selectively** and answers **honestly** — and "honestly" is enforced in code, not requested in a prompt. It runs two rules that pull against each other, and the engineering is in how they're reconciled.

RULE 1 · BE BOLD

Don't miss opportunities

A stretch role that rejects you costs nothing. A good role you never saw is a real loss. So false negatives are treated as worse than false positives — a senior title with a genuine AI or automation signal gets collected, not discarded.

RULE 2 · BE HONEST

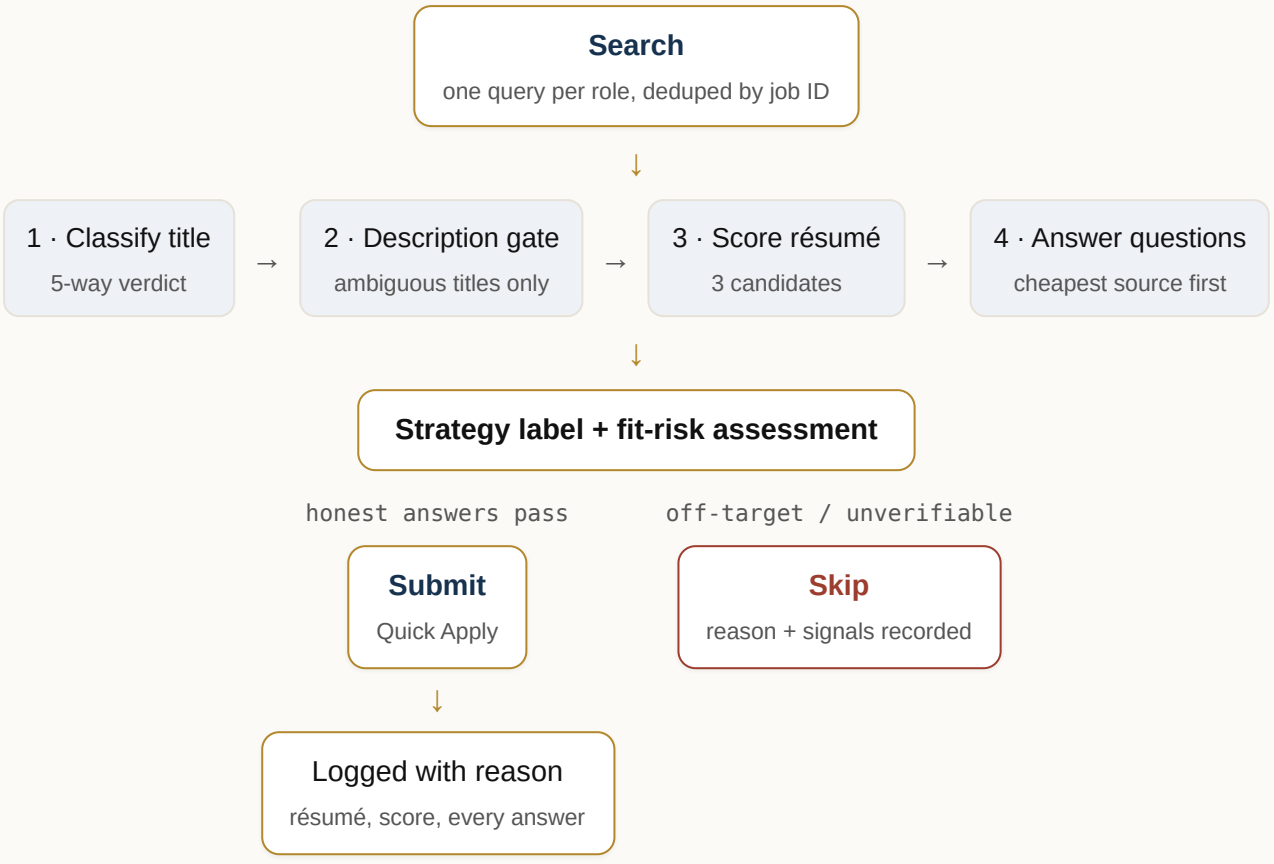
Never overclaim

Round experience down, never up. No certification, clearance or language is claimed unless it's confirmed in the profile. No job title is quietly reframed into a better one. If honest answers disqualify you, that is the correct outcome.

The agent does not protect the applicant from **rejection**.
It protects the employer from **inaccurate answers**.

02 How a listing becomes an application

Every job runs through a fixed funnel. Most listings are eliminated on the title alone, for free. Only genuinely ambiguous ones are expensive enough to warrant reading the description. Every decision — including every rejection — is written to the log with the reason and the signals that produced it.



Nothing is discarded silently — a skip is a logged decision, not an absence.

TWO GATES A title lands in one of five buckets: **clear positive** (apply), **hard exclude** (never — director, chief, trades, clinical), **clear negative**, or one of two **ambiguous** grades. Only ambiguous titles cost a description read. The rescue rule matters most: a seniority word like *Lead* or *Manager* is **not** disqualifying if the title also carries a real target signal — *AI Enablement Lead* survives, *Lead Accountant* does not.

SOFT vs HARD In the description gate, negatives are weighted by kind. *Construction* is a **soft** negative — it can't cancel explicit IT signals, so an *IT Site Coordinator* on a building site still passes. *Clinical*, *10+ years* or *PhD required* are **hard** negatives and always block. This distinction was added after real listings were being dropped for the wrong reason.

03 What keeps it honest

Every claim below is a rule in code with a regression test behind it, not an instruction in a prompt. A prompt can be talked out of its principles; a guard clause cannot.

It refuses to claim credentials

Security clearances, certifications and working-with-children checks are hard-blocked unless confirmed in the profile — even if the form has pre-filled a "yes".

It rounds down, never up

On any years-of-experience question the agent takes the lower bracket. Two years of IT support is answered as two, not "2–5".

It distrusts the form's memory

SEEK pre-fills answers from past applications. Those are re-checked against 208 overrides and corrected before submission — the most common source of accidental overclaiming.

It won't reframe a job title

A Bookings Officer who used IT systems daily is not an IT Support Officer. The agent describes the role that existed, not the role that would score better.

It only speaks the languages it speaks

Any language not listed in the profile defaults to the lowest proficiency option — closing a real bug where forms pre-filled "Native or Bilingual" for languages never spoken.

It states its own confidence

Each application is labelled — target fit, strategic stretch, worth a shot, or low-confidence default — so a weak application is visible as weak rather than hidden among the strong ones.

It explains every rejection

A skipped job records its classification, the signals that matched, the description score and the threshold it missed. Skips are auditable, so bad filtering is findable.

It can be told it was wrong

202 regression tests pin the behaviour, and `--recheck-skips` replays every past rejection through improved logic — so a fix reaches the jobs it already got wrong.

04 Under the hood

A single-module Python agent driving a real browser through Playwright. Claude is a component inside it, not the architecture of it: the model is called only when the local cascade has genuinely run out of answers. That cascade is the distinctive engineering — three free sources are exhausted before a single API call is made, and one guard sits across all four.

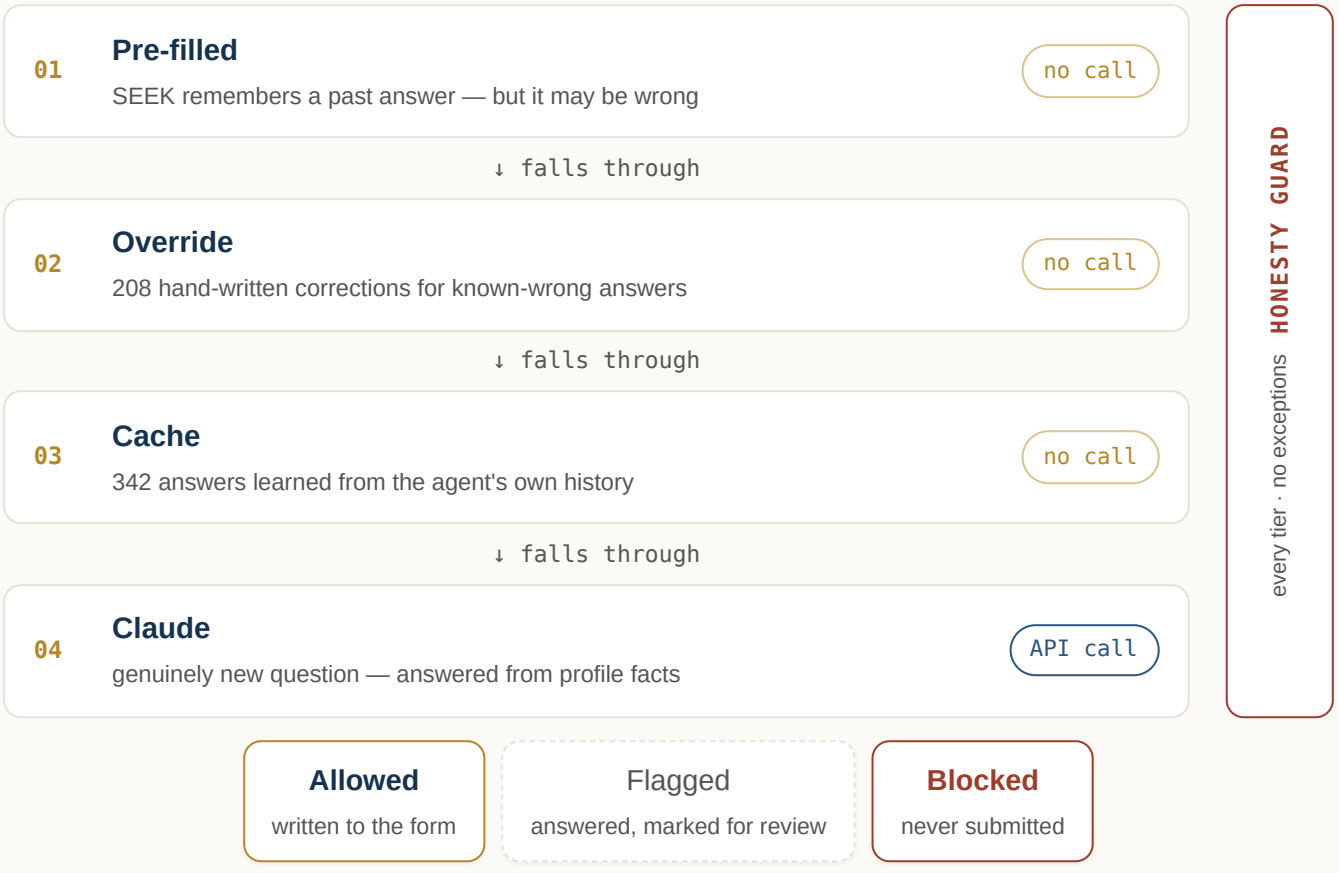
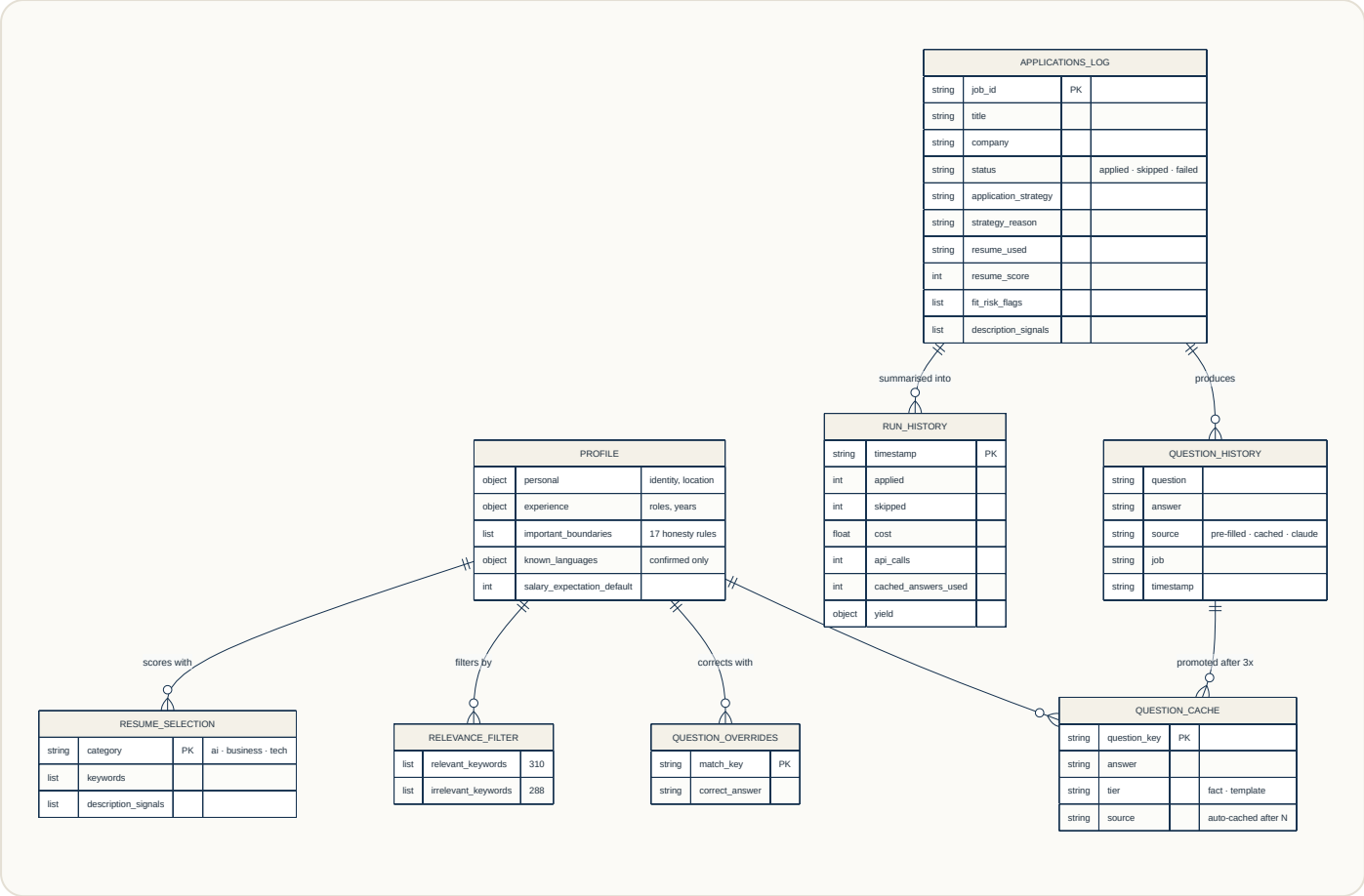


Figure 1 — The four-tier answer cascade. The guard spans every tier, including the free ones: a cached answer is not trusted merely because the agent wrote it.

COMPONENT	WHAT IT DOES
`classify_title`	Returns one of five verdicts for a job title, splitting hard excludes (never rescued) from seniority words (rescued when a real target signal is present).
`check_description_relevance`	Second gate for ambiguous titles only. Scores positive and negative signals, weighting soft negatives below explicit technical ones.
`choose_resume`	Scores all three résumés — AI, business systems, and technical — against title, description and search intent, and records the runner-up and margin for auditing.
`validate_answer_safety`	The central guard. Runs on every answer from every source and returns allow, flag or block. Cached answers are blocked outright on credential claims to force fresh evaluation.
`auto_cache_from_history`	Promotes an answer to permanent memory after three consistent uses — the mechanism that drives API cost toward zero over time.
`determine_application_strategy`	Combines title verdict, fit risk and résumé confidence into a plain-English label and reason for every application.
`already_applied`	Deduplicates by stable job ID across all runs. Description rejections are final unless explicitly replayed, so the agent never re-pays to reconsider the same listing.
`rate-limit tracker`	Tracks applications per hour and per 24 hours, delays between submissions, and stops the run after consecutive platform errors.

05 Data model

There is no database. Memory is four JSON stores on disk, which is the right shape for a single-user agent that must be readable, hand-correctable and diffable in Git. The relationship that matters is the loop: **question_history** is distilled into **question_cache**, so the agent's own past is what makes its future cheap.



PK key field **one ──< many** (one log produces many questions)

JSON document stores, not tables

Figure 2 — The four local stores and the learning loop between them.

> **WHY JSON** A relational store would be the reflex choice and the wrong one here. The profile is edited by hand constantly — a wrong answer is fixed by opening the file and correcting it. Keeping memory as readable documents means every override is reviewable in a diff, and the honesty rules stay legible to the person accountable for them.

06 Outcomes

330	2,059	367	202
LIVE APPLICATIONS	LOGGED DECISIONS	DISTINCT EMPLOYERS	AUTOMATED TESTS
69%			
QUESTIONS ANSWERED FREE			

The application log holds 2,059 job decisions: 474 applications (330 live, 144 in dry-run), 1,543 skips and 42 failures — every one carrying the reason it was made, across 367 distinct employers. The agent has classified 1,547 distinct job titles and eliminated the overwhelming majority on the title alone, for free.

The economics are the point. Of 1,694 employer questions encountered, **69% were answered with no model call at all** — from the form's own memory, from 208 overrides, or from the 342-entry cache the agent distilled out of its own history. Only the genuinely novel 31% reached Claude. Of the runs carrying cost telemetry — 23 runs and 167 of those applications — total spend was **$0.59**, about a third of a cent per application. That rate falls as the cache grows, because every question answered three times the same way stops being a question.

The safety record is the part worth defending: no certification, clearance or language has been claimed that wasn't in the profile, because the guard runs on every answer from every source — including the ones the agent wrote itself.

> **SCOPE** Two different windows, deliberately not blended. The **330 live applications** come from the full application log; the **$0.59 / 167 applications** figure covers only the 23 runs recorded after per-run cost telemetry was added. Dividing total spend by total lifetime applications would flatter the number by roughly half — so the denominators are reported separately rather than averaged into one better-looking statistic.

Python · Browser automation (Playwright) · LLM application & prompt design · Decision-cascade architecture · Cost-aware API usage · Rules-vs-model trade-off design · Automated regression testing · Structured logging & metrics · Data modelling (document stores) · Git

Jira · Confluence

Personal project by Mason Tatafu · figures generated from the agent's own logs (`applications_log.json`, `question_history.json`, `run_history.json`) and a test run of the live suite. The agent runs locally and applies only on Mason's own behalf; it holds no credentials beyond a saved browser session and an API key kept out of source control.