DSA · DIGITAL STUDENT ASSISTANT

How DSA Works

Engineering a supervised LLM tutor: the pipeline that builds a course, the turn that withholds the answer, and what it takes to run it

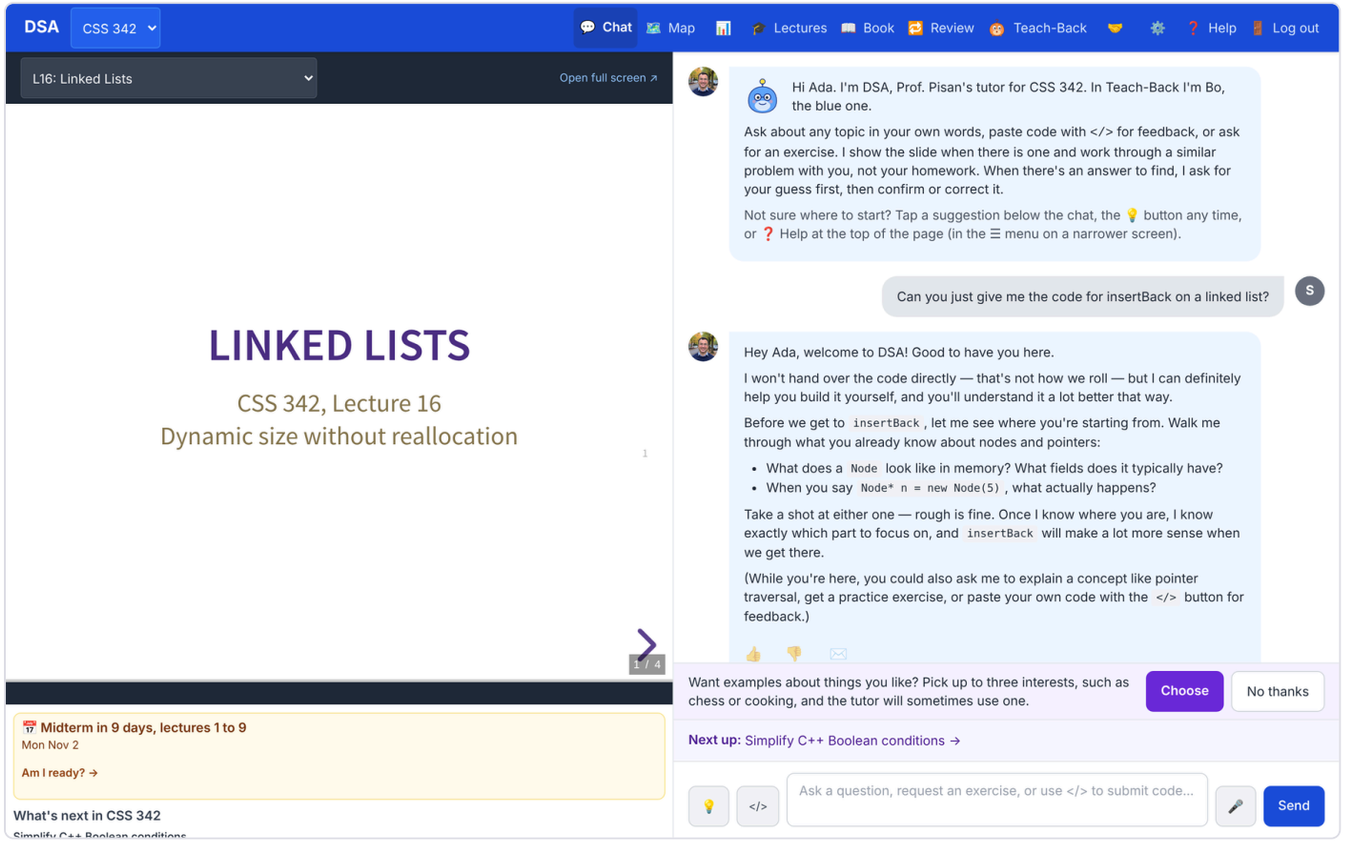
Yusuf Pisan

University of Washington Bothell, Computing & Software Systems

pisan@uw.edu · digital-student-assistant.com

THE SYSTEM

A course tutor that answers from the course, and will not do the homework



- The tutor for CSS 342 and 343, our C++ data structures sequence. In production with students since a June 2026 pilot.
- **Two deliverables in one repo:** an offline pipeline that builds each course, and a FastAPI app that serves it through a supervised Claude tutor.
- Around the chat: a topic map, mastery checks, spaced review, find-my-bug, Teach-Back, exam readiness, and instructor pages.

THE ONE DECISION EVERYTHING ELSE FOLLOWS FROM

The model writes the reply.
Ordinary code decides what the reply may contain.

Why not a better prompt

A capable model asked to tutor gives the answer by default. Prompt-only withholding leaks at scale: CS50's duck put code blocks in about 22% of replies despite a no-solutions prompt.
Liu et al. (SIGCSE 2025); Macina et al. (2023)

What that buys

A decision you can table-test, a boundary student text cannot cross, and a definition of compliance you can measure: the reply stayed at or below the ceiling code computed.

Roadmap

1. **Building a course**

   sources in, eight course-stamped collections, generated content you can check

2. **The supervised turn**

   ten stages, who sees what, the ceiling, the detector and the judge

3. **The learner model**

   evidence, mastery, decay, and how they set the ceiling

4. **Running it**

   deploys, money, security, data rights, operations

5. **Keeping it correct**

   tests that read the code, simulated students at $0, how it was built

6. **Lessons and open problems**

Opening

AT A GLANCE

A Python monolith on one container

Web	FastAPI, Jinja2, Alpine.js, precompiled Tailwind; replies stream over Server-Sent Events
Data	SQLAlchemy 2 and Alembic; Postgres on Supabase in production, SQLite locally; 59 tables
Retrieval	ChromaDB on disk, Chroma's ONNX all-MiniLM-L6-v2 for both building and querying
Models	Sonnet 4.6 writes replies; Haiku 4.5 classifies, plans, judges and grades; Sonnet re-judges nightly
Hosting	One Docker service on Render with a persistent disk; manual deploys
Offline	Whisper, python-pptx, pdfplumber, ElevenLabs narration, the `claude` CLI

88K
lines of app Python

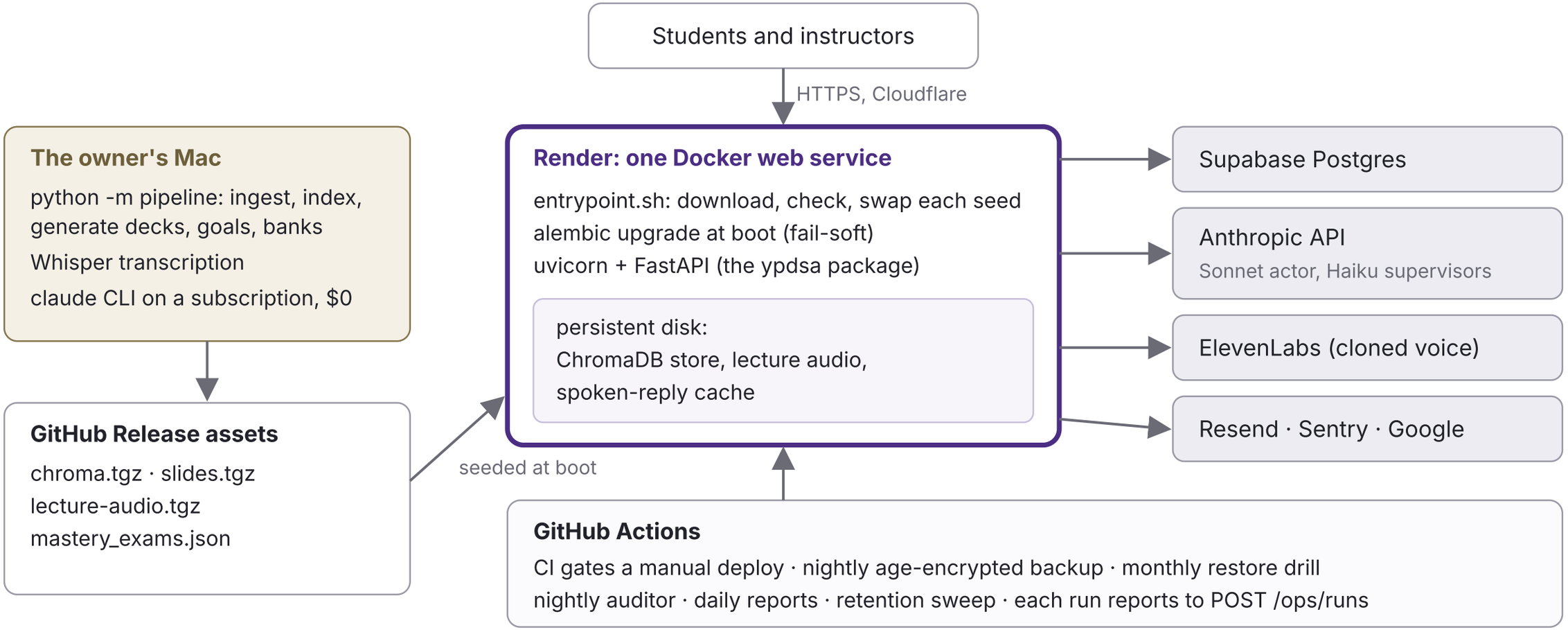
166K
lines of tests, 1.9× the app

10.5K
lines in the pipeline, 29 commands

30
pedagogy flags, each a kill switch

SYSTEM CONTEXT

Built on a laptop, shipped as release assets, served from one container



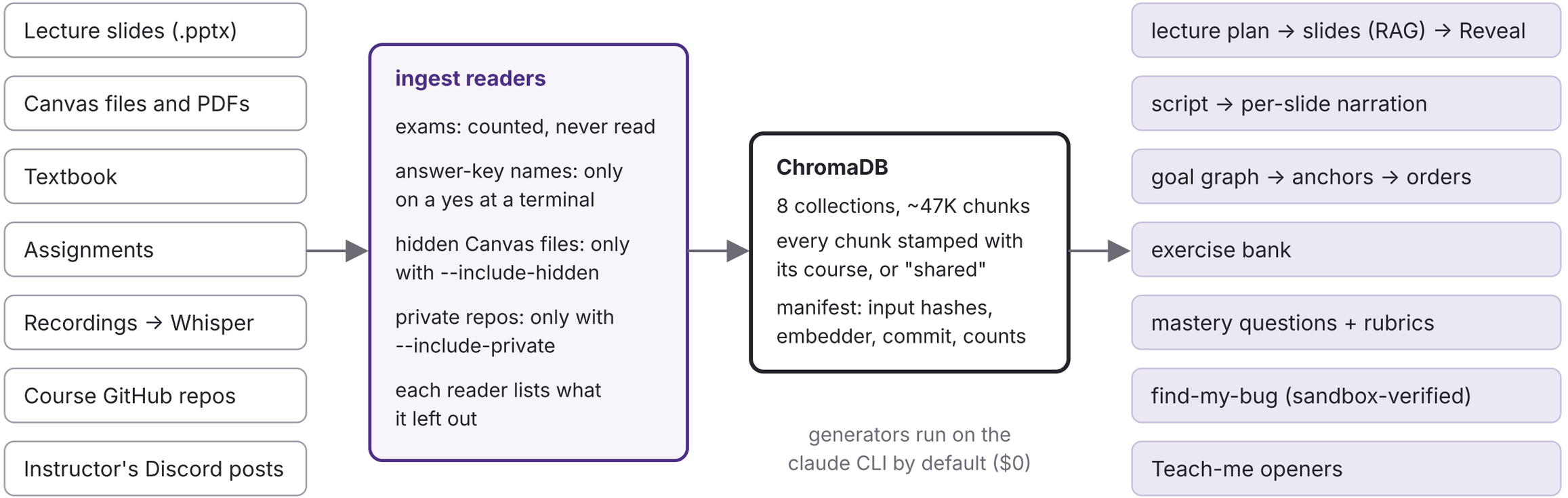
PART 1

Building a course

Sources in, eight course-stamped collections, and generated content you can check.

THE OFFLINE PIPELINE

Sources in, eight collections out, a course generated from them



The collection list lives once, in `ypdsa/kb.py`; a test holds the indexer's builders to it.

TWO WALLS

Answer keys stay out at ingest, and stay hidden at retrieval

Wall 1: ingest

- Exams are counted, recorded as a number, and never read.
- One name rule, `answer_name`: solution, answer, soln, quiz, midterm, exam, final, key, with camelCase split ("MidTerm", "hwkey") and exclusions ("keyboard", "monkey").
- An instructor's Canvas token sees everything, so files students cannot see now (hidden, unpublished, locked, future unlock) are dropped unless asked for.
- Students' private GitHub Classroom repos are skipped; Discord keeps only the instructors' posts, with no author field.

Wall 2: retrieval

- Anything that did get indexed and looks like a solution or key is filtered out of `find_topic`.
- A search keeps its course's chunks plus the shared ones, but only where the collection is stamped: an older, unstamped store is searched course-blind, never emptied. Production's store is that older kind until the rebuilt one ships with CSS 430.

And the repo itself

- CI fails on paths that look like student records, any file over 10 MB, a stray secret, or a jump of 200+ files without a stated reason.

WHAT GETS GENERATED

One course is a registry entry plus a set of checked files

40
lecture decks for 342 and 343, with per-slide narration in the instructor's cloned voice

216
learning goals with 236 prerequisite edges, across three courses

325
exercises: concept, trace, code and debug

~8
mastery questions per goal, each with a rubric

32
find-my-bug programs, each verified in the sandbox

216
pre-written "Teach me" explanations, served with no model call

47
narrated book pages, built with Quarto into the image

1
registry, `curriculum/courses.json`; no app module names a course

Content counts from the repo, not usage. Decks, narration and the mastery bank are gitignored and ship as release assets.

GENERATED CONTENT YOU CAN TRUST

Facts, not claims

Artifact	How it earns its place
Find-my-bug item	Kept only if the fixed program passes every test with ASan and UBSan clean, the buggy one compiles and fails the exposing test, and the two differ only on the stated lines. Ids are never reused.
Mastery bank	Each goal's pool must pass a structural check (4 to 12 questions, a rubric each, no duplicates, at least one concept question) or it is not saved and the command exits 1. Question ids are a hash of the text.
Narration	A script records the hash of the slides it narrates; new slides move the old narration to `_stale/`, never deleted.
Voice profile	Analyzers *propose* into `proposed/`; nothing overwrites the curated files.
Every write	Atomic: temp file, fsync, `os.replace`. A test scans the pipeline for plain writes.
Retrieval	A golden set runs through the real `find_topic` weekly in CI, and checks no other course's decks leak into a course's results.

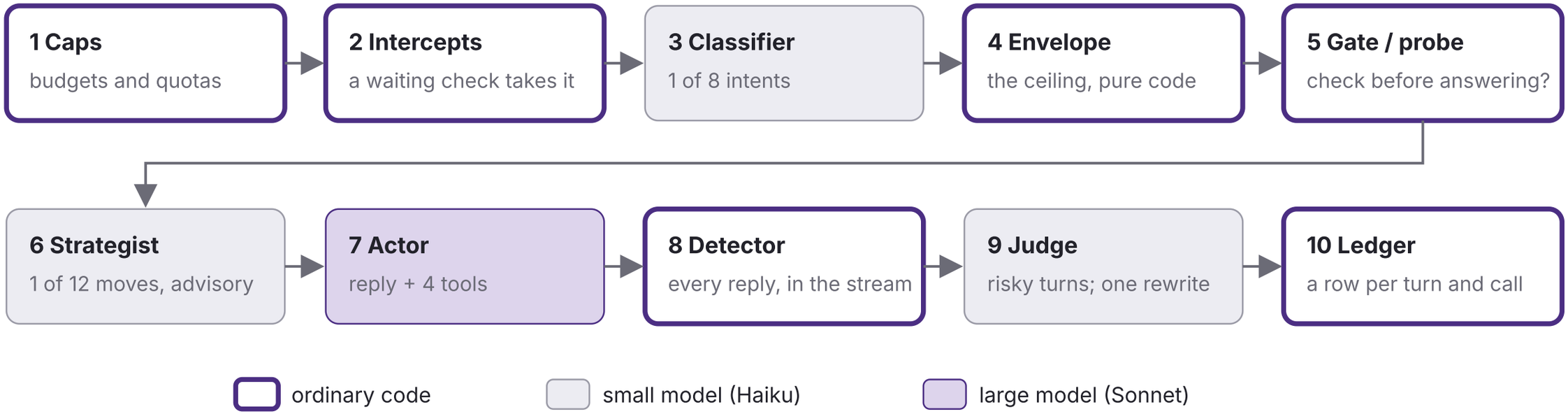
PART 2

The supervised turn

Ten stages, who sees what, the ceiling, the detector, the judge and the ledger.

ONE PATH, `run_turn`

Every message goes through the same ten stages



- Nothing else may import a stage, and a test reads the code to make sure. One turn per student at a time; the message commits before the actor runs, and no turn holds a database connection while it waits on a model.

INFORMATION FLOW IS THE DEFENCE

Who sees what

Stage	Student's words	Learner state	The draft	What it can decide
Classifier	framed	–	–	A label: 1 of 8 intents, up to 3 goals from the course's catalog, "has an attempt". No policy.
Envelope	never	yes	–	The ceiling. Pure function, no model call, imports nothing that carries prose.
Strategist	never	facts	–	One opening move. Its signature cannot accept the words; it cannot raise the ceiling.
Actor	framed	rendered	writes it	The reply, inside the contract. Writes no state.
Detector	–	ceiling	yes	Cuts code. Deterministic.
Judge	never*	envelope	redacted	Allow, one rewrite, or block.

* Except the student's own code, for the review of a long block. So a student can try to instruct the actor, but never the components that bind it. Pattern: Willison's dual-LLM (2023); injection via retrieved text: Greshake et al. (2023).

UNTRUSTED TEXT

Every string a student or instructor wrote is framed as data

```
# agent/framing.py
frame("student_message", text)
  → <student_message> ... </student_message>
     # every frame's closing tag inside the text
     # is neutralised: <\/student_message>

head_tail(text, limit)
     # keeps the head AND the last 1,000 chars,
     # so padding cannot push the real ask out of view

safe_name(display_name)
     # letters, digits, spaces, ' . -, at most 40
```

- **18 frame tags:** student message, answer and code, transcripts, drafts, the rolling summary, class materials, exam names, run results, a TA's message, and more.
- **Placement:** the rolling summary and sandbox results ride in the user turn, never the system prompt.
- **Instructors too:** an exam name or uploaded material reaches every enrolled student's prompt, so it is framed like student text.
- **INJECTION is an intent.** It never raises the ceiling, and it forces a three-sample judge.

THE CEILING

Policy as data: a ladder, four bands, and seven steps in order

H0	acknowledge and encourage
H1	restate or clarify the problem
H2	point to the concept or slide
H3	a leading question, next step
H4	the approach in words, no code
H5	a worked example, *different* problem
H6	pseudocode, key lines `___`
H7	full solution (explain mode only)

Bands: mastery ≥ 0.85 → H2; ≥ 0.70 → H3; ≥ 0.40 → H4; below → H5. Less shown, more scaffolding.

1. **Base:** the most permissive band among the goals in scope. No evidence on one goal means H5.
2. **Code floor:** an attempt, or a question about the student's code, gets at least H4. Never on a degraded classification.
3. **Escalation:** an impasse or a granted "Show me more" adds one, capped at H6, counted from the band and floor, so it cannot ratchet.
4. **Exam review:** lifts to H5 in a review window, only if every goal is in range and mastered and the turn has no code.
5. **Lockdown:** a goal under a pending mastery check caps the turn at H1.
6. **Explain mode** (admin, per course) reaches H7, never past a lockdown or a graded assignment.
7. **Conduct restriction:** H1, applied last, so nothing lifts it.

Solution demands and adversarial intents never raise *and never tighten* the ceiling: route kindly.

INSIDE THE CEILING

A ceiling on *how much* is not enough. There is also *what*.

The conclusion rule

Below H7 the tutor may not state the conclusion (a complexity class, a base case, the buggy line, "your code is right") until the student commits, as a trusted flag records.

The actor and the judge read the **same withheld words** from one constant, so the two cannot drift apart:

```
WITHHELD_AT_H4 = "the specific bug, the exact
  wrong token, or the corrected value"
```

Moves, fading, and the cold start

- The strategist picks 1 of 12 moves (elicit a prediction, a worked analogy, a completion problem…). A move that shows content needs a minimum rung.
- Worked examples fade to completion problems, then full problems, from the last two graded scores.
- One goal with no evidence pins a turn at H5. So ordinary use has to *produce* evidence: hence the predict-first probe and offered quick checks.

THE ACTOR

A cached prefix that names no course, and a small per-turn contract

System block 1: cached, 1-hour TTL

preamble and the four tools

persona: 3 files, the instructor's voice

instructor profile: 5 files (flow, patterns, examples)

the H0 to H7 legend and the move vocabulary

PERSONA_OVERRIDE: where the tutor differs

142,663 characters, about 40K tokens (est.)

System block 2: uncached, this turn

student state; "THIS TURN: respond at hint level ≤ H4"

messages: summary, last 8 verbatim, this one (framed)

- **Four tools:** `find_topic` (moves the slide panel), `get_lecture_outline`, `get_exercise`, `evaluate_code` (a 0 to 4 score and one hint, never corrected code). The envelope filters which ones the actor gets.
- **Bounded loop:** at most three tool rounds. Tool inputs are not trusted; a call stays inside the courses the student may open.
- **One prefix for every course:** it names no course, so one cache entry serves them all.
- **Memory:** eight messages verbatim; older ones fold into a rolling summary after the reply, in a task of its own.

COST IS ARCHITECTURE

Caching the prefix: a 1-hour TTL, matched to how students read

Per actor call, ~40K-token prefix	Cost
Uncached input at $3/M	$0.12
Cache read at $0.30/M	$0.012
1-hour cache write at 2× input	$0.24

Arithmetic on list prices, not measured spend. The write pays for itself on the second cached call within the hour.

- **Why not 5 minutes:** students read for longer than that between messages, so a 5-minute entry had always expired.
- **Warm-up:** opening the chat page sends a `max_tokens=0` request that writes the cache, skipped while the entry is alive.
- **Order matters:** cached block first, varying block last, tools declared identically on every call, or the cache never hits.
- **Other levers:** small models supervise; the judge runs only on risky turns; a gated turn spends nothing on retrieval.

DETERMINISTIC FIRST

The detector runs on every reply, inside the stream

- **Cuts** any C++ block over 4 lines, and any C++ at all at H4 and below (signatures and blanked templates too). At H5 and H6 a long block is held for the judge's code review instead.
- **Finds code** in fences, tildes, indented blocks, raw `<pre>`, runs of single-span lines, bare code at the margin, and a block that closes braces an earlier one opened.
- **Decodes** base64 runs and flags the ones that decode to C++.
- **Exempt:** pseudocode with `___` blanks, four lines or fewer, inline spans in prose.

Streaming without leaking

On routine turns prose streams at once (risky turns are buffered for the judge). Inside a code block, a `StreamGate` holds tokens until the block closes, then emits it or the line *"I'll keep the full code out of this one."* A base64 run split across chunks is held too.

About 1,280 lines, pure, no model and no network. The policy, detector and judge tests together are about a thousand cases.

MODEL SECOND

The judge reads the draft and the contract, never the student

- **When:** solution demands, exam leaks, injections, lockdowns, code-adjacent turns at H4 or below, every escalated turn, graded-assignment turns, every degraded classification. Routine concept chat skips it.
- **What it sees:** the envelope, the ladder, the cited slide's title, the redacted draft.
- **Verdict:** allow, revise or block, with a rule from a fixed list. High-risk turns take three samples and a majority; ties go strict.
- **Revise once:** the actor rewrites with the violation named; failing that, a templated pointer to the slides.

A real verdict, from a local run this week (every model call through the `claude` CLI, no knowledge base, a demo student, no production data):

> "The draft shows a complete, compilable function solving a different problem (sumList), which is compliant at H5 as a worked example, but then immediately applies it to the student's own task by naming the exact bug in their insertBack code..."
>
> rule: `code_reveal` · verdict: revise · detector cut: yes

The worked example was allowed. Pointing it at the student's own bug was not. The rewrite failed too, so the student got the templated pointer to the slides.

FAILURE IS A DESIGN INPUT

Fail closed where harm is asymmetric; fail open where silence is the harm

Closed

- No judge verdict on a high-risk or escalated turn: the reply is withheld ("could not be checked; try again in a minute").
- A degraded classification: intent becomes OTHER, no code floor, no lifts, a strict detector, a forced judge.
- Graders never auto-pass a check they could not read.
- A circuit breaker on supervisor failures: in a full outage every reply is withheld within about 20 seconds, and the operator is emailed.

Open, or soft

- A routine turn when the judge is down: the tutor still answers, under the detector.
- Maintenance mode on a database error: it does not lock everyone out.
- The strategist and the rolling summary are advisory: any failure means no move, no summary.
- Boot never stops on a missing seed or a migration warning.

EVERYTHING OBSERVABLE

The ledger: a row per turn, per model call, per graded answer

```
# turn_events, three turns from the same local run
kind=chat  intent=SOLUTION_DEMAND  rung_used=5
      verdict=allow   judged=true  tool_rounds=1
kind=chat  intent=ATTEMPT          rung_used=5
      verdict=allow   judged=true  tool=evaluate_code
kind=chat  intent=QUESTION         rung_used=5
      verdict=revise  detector_cut=true
      judge_rule=code_reveal  code_review_held=true
# plus one row per classify, strategist and judge call,
# each priced, with its payer and turn id
```

- `rung_used` is the ceiling **enforced**, not the rung the reply reached. Measuring the latter is an open problem.
- One function, `ledger.record_call`, prices and writes every model call's row. It replaced about 18 hand-written sites.
- The ledger feeds the admin's Supervision tab, the daily digest, the cost report and the nightly auditor.

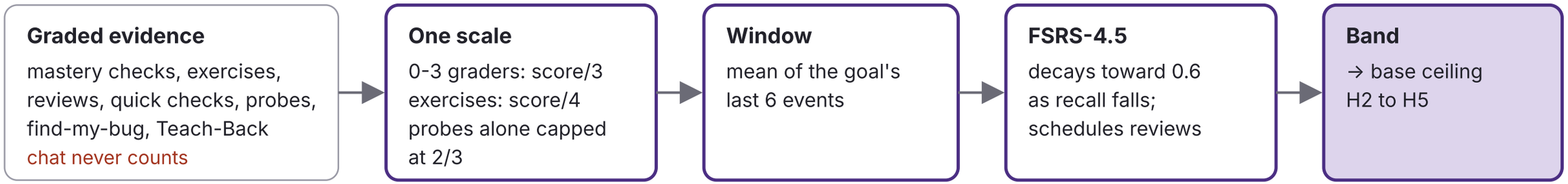
PART 3

The learner model

Evidence, mastery, decay, and how they set the ceiling.

FROM EVIDENCE TO CEILING

One grade scale, a window of six, a forgetting curve



Status only moves up: locked → unlocked (0.6) → mastered (0.85 with 4 events, or a check passed 2 of 3) → confirmed (two successes 16 hours apart)

A well-practised goal (high FSRS stability) barely decays; forgetting raises the ceiling back toward H4 and H5 on its own. Mastery is sticky by design: a bad streak shows through decay and the review clock, not by relocking.

Activity data (page views, slide dwell) only *triggers* offers, never evidence or a ceiling input. Only a student's first answer per find-my-bug item counts.

GETTING EVIDENCE WITHOUT A TEST DAY

Gates, probes and offered checks

Prerequisite gate

An attempt on a lecture with an unmastered earlier prerequisite may wait behind three questions sampled from that goal's pool of about eight. Two of three passes.

While it waits, the goal is locked down: H1, no retrieval, no exercises, no escalation, a three-sample judge.

Predict-first probe

A conceptual question on a goal with no evidence may first get one prediction question, then the answer at the new ceiling.

At most once per conversation and per goal in 14 days. A probe alone never sets a goal below H4.

Offered quick checks

After a failed exercise, a check with no tries left, or a mostly watched lecture. They set review timing, never the map.

Only one module decides who gets a probe, one which goal, one which offer.

PART 4

Running it

Deploys, money, security, data rights and operations, for one person.

DEPLOYS

Manual, gated, and a boot that never stops on an external

- **Manual deploys:** a workflow that refuses a commit the tests have not passed, and drives a maintenance page over the swap.
- **Seeds as release assets:** the store and audio land once on the persistent disk; the decks and the mastery bank on every boot. Free egress, no object store.
- **Fail-soft boot:** each seed is downloaded with retries, unpacked to scratch, checked, then swapped in, with a marker written last. A failure keeps the previous copy and is reported once the app is up.
- **Migrations at boot:** every revision after the baseline is guarded. A column is dropped only one deploy after the models stop mapping it, so a rollback still works.
- **Restarts:** a reloaded page reattaches to its running reply; a turn a deploy cuts off is marked interrupted; a 502 shows "restarting" and never eats what the student was typing.
- **The image:** non-root, the embedding model baked in, and **no compiler**.

MONEY

Three caps, three payers, one function that decides

Caps

- A daily turn limit per student and a monthly cap for the class, each with an exam-week value.
- Per-student dollar quotas by day, week and month, with one extension per period.
- Students see "about N questions left", never dollars; every refusal says when it lifts.

Who pays, in order

1. The site's key, within the caps
2. An instructor's key, for their own rosters, in that course, chat only
3. The student's own key, with their own ceiling

`usage.decide`, inside `run_turn`, alone decides. Keys are encrypted per purpose and decrypted only for a turn.

Development-time estimate, June 2026: about $0.02 to $0.05 a turn, planned at $0.03. Clearing one environment variable is the kill switch.

SECURITY

Small, boring defences, each with a test

Content Security Policy

Enforced on every response; a fresh nonce per page for scripts, no inline handlers, libraries vendored. HTML only in templates, JSON through `|tojson`.

Request guard

Before any route: a write from another site is refused (this closes login CSRF), and every path has a body ceiling, checked before a byte is read.

Sign-in

Google, or a 6-digit code. UW's mail scanner opens every link on delivery, so the emailed link opens a Continue page instead of signing in.

Read-only view-as

Staff can see a student's pages; every write is refused, writing GETs are skipped, and the right is re-checked on every request.

The `claude` CLI, locked down

One launcher: every tool off, no API key in its environment, an empty scratch directory, safe mode. A test fails on any other launch.

What Postgres refuses

SQLite is made to refuse it too in tests; an input sweep posts bad values into every field of every student route.

DATA RIGHTS BY CONSTRUCTION

Export, delete and forget, with one producer each

- **Export:** one function builds every data package, and its audience has no default. A table in the erasure scope with no export decision fails a test.
- **Delete:** the package is sealed with `age` *before* any row goes. Tombstones outlive every backup, so a deletion can be re-applied after a restore.
- **Forget:** a daily sweep anonymizes identifiable records a year after the term, keeping de-identified telemetry under a pseudonym.
- **Minimal by construction:** activity events carry ids and counts, never text; interests and reply language are fixed codes; logs name students by number; Sentry is scrubbed.
- **Never in a prompt:** a roster. **Never held:** an instructor's Canvas token. **Never sent:** grades, either way.
- **Class figures:** no percentile under five students; staff out of every median.

OPERATIONS

Backups you have restored, and jobs that report in

- **Nightly `pg_dump`**, encrypted with `age` to an offline key, kept 21 days, plus an off-site copy. The job fails rather than store a readable dump.
- **A monthly restore drill.**
- **An ops ledger:** every scheduled job reports each run; a lateness check catches a job that never started, which an in-run failure email cannot.
- **Monitoring:** an outside uptime probe, Sentry, an email on a burst of tutor failures, and a row for every error page a browser gets.

The single-instance assumption

Process memory holds rate limits, the per-student turn claim, reply buffers a reloaded page reattaches to, grading claims, the supervisor breaker and more, in about two dozen modules.

All of it is listed in one place, and it moves to Postgres before a second instance runs. New in-process state must join the list.

PART 5

Keeping it correct

Tests that read the code, simulated students at $0, and how it was built.

TESTS

Most invariants are a test, and many tests read the code

9,627
collected test cases, network-free

600
in headless Chromium, phone width included

42
test files that parse the code with `ast`

PG 17
a required CI job against real Postgres

"One X" rules, each enforced by reading the source:

- only `agent/state/` writes learner state
- only `actor_client.py` calls the actor model; no other Anthropic client
- only `cli_launch.py` starts the CLI
- no async route touches the database off the threadpool
- every environment variable is in the manifest
- no non-atomic write in the pipeline

Plus mypy on every router, the agent, scripts and the pipeline, with no override; a clock scan that runs date-bound tests a week early; random test order.

MEASURING THE MECHANISM

Simulated students at $0, a stronger model at night

The persona simulator

- 29 literature-grounded student cards (the recursion loop-modeller, the help-avoider, the social engineer, the exam leaker…) at seeded mastery bands.
- They drive the **real** `run_turn`, every model call answered by the `claude` CLI on a subscription: building an API client raises, and each conversation gets a throwaway database.
- `compare` runs two arms that differ by one flag.

The nightly auditor

- Sonnet re-judges every risky turn of the last day plus a sample of routine ones, and reports under-blocking ("the dangerous direction") and over-blocking ("pushes students to unguarded ChatGPT").
- It never blocks the live pipeline.

Flags decided by experiment

Judging every code turn is kept off in production after a three-repeat comparison showed more revisions and no safety gain.

Caveat: simulated students learn faster than real ones. On the limits of simulated students: Scarlatos et al. (ACL 2026).

HOW IT WAS BUILT

One developer, many AI coding sessions, and a rulebook

- **Claude Code sessions** wrote most of the code: in the week the clone covers, 176 of 316 commits carry a Claude co-author line.
- **`CLAUDE.md` is the rulebook:** 48 one- or two-line invariants, each with a long-form section, updated in the same commit as the behaviour.
- **One topic per pull request,** a change note per PR, dated handoff documents so a fresh session can continue, and numbered owner decisions.
- **A pre-deploy review in nine slices**, each read against the invariants: about 209 findings. Four reviewers found the same top bug independently.
- **A follow-up by a read-only workflow of 34 agents:** sweepers, verifiers, a skeptic, re-checkers and a planner, producing 15 phases.
- **The owner alone** writes to production, approves paid calls, deploys and flips switches.

> The tests that read the code are what let a large number of AI-authored pull requests land without eroding the architecture.

PART 6

Lessons and open problems

WHAT I WOULD KEEP

Six lessons

1. **Decisions belong in code, not prompts.** The prompt states the rule; a pure function computes it; the detector and judge enforce it.
2. **Keep untrusted text away from the decision makers.** Information flow beats clever prompting: the policy core, the strategist and the judge never read the student.
3. **Deterministic first, a cheap model second, a stronger model offline** to measure the cheap one.
4. **Fail closed where harm is asymmetric, open where silence is the harm.** Write down which is which.
5. **One writer, one launcher, one decider, enforced by tests that read the code.**
6. **Facts, not claims, for generated content.** A bug item exists only if the sandbox shows it; a bank is saved only if it passes its check.

WHAT IS STILL OPEN

The problems I have not solved

Measuring the rung reached

The ledger records the ceiling enforced, not how much a reply actually gave away. Prose that states the answer is a known blind spot of the detector.

Validating the judge

Compared so far only with a stronger model. Blind sheets for human labels exist; the labels do not yet.

Running student code

Off in production: exercises are graded by a model's rubric. A sandbox is built, contained on Linux, and used offline only.

Scale and people

One instance, one persona, one maintainer, no LTI. And no controlled study of whether students learn more.

Questions

Yusuf Pisan
pisan@uw.edu

Companion talks: *Hints, Not Answers* (the pedagogy), *Put Your Course on DSA*, and *Teaching an LLM Tutor to Withhold the Answer* (Northwestern, October 2026)

Slides at pisanorg.github.io/yusuf/talks/202610DSA-HowItWorks