

Rubrica: privacy-first AI exam grading

Software description for version 0.1.10 · Zenodo report, revision 8 · exported 2026-10-05

What this document is. The project README for Rubrica's open-source core, exported as the software description for the version 0.1.10 report. Version 0.1.10 is the current pilot build (October 2026): the same grading backend described here, packaged as a standalone, signed and notarized macOS application for Apple Silicon Macs on macOS 12 or later (a Tauri shell around the Flask backend, with a React interface; on macOS 14 or later it includes the Ollama runtime that reads cover pages locally). The grading prompt and the scoring rules are unchanged from version 0.1.2, the first pilot build (late August 2026), for a model response that is not empty and gives each question once; model versions, page rendering and rubric formatting are unchanged too. Version 0.1.10 changes how two other kinds of response are handled: when a question is returned more than once (labels that differ only in spacing, capitals or punctuation, such as "Q1" and "Q 1", count as the same question), the first entry is kept and the exam is flagged for review, and a total returned with no per-question scores is discarded (each question is set to 0, marked not graded and flagged, or the exam is left ungraded). An instructor's score edit also settles a boundary average that has not been resolved: the total becomes the one derived from the question scores, with rubric normalization kept. An average the instructor has already accepted stays offset by the same amount through later edits. The other changes in versions 0.1.3 to 0.1.10 are platform support, connection security in the packaged app, the review and first-run interface, and (in 0.1.6) how rubric enhancement matches the AI's proposed criteria to questions before the instructor reviews them, and (in 0.1.8) a local OCR runtime included in the app, sharing an enhanced rubric between graders, and a corrected ICC calculation in the app's agreement reports, and (in 0.1.9 and 0.1.10) search on the exam lists, report email that runs in the background and holds back a report that may already have been sent, review actions that apply only to the grade shown on screen, and copies of saved workspaces to a cloud folder that are off unless the instructor turns them on. The setup section below describes running the core from source.

Source availability and licensing. This record contains documentation and validation evidence, not source code. Rubrica is open-core: the core is licensed under the GNU Affero General Public License v3.0 and the commercial modules are proprietary. The AGPL core is the Flask backend, the scoring package, and the Jinja templates; the React interface, Tauri shell, sign-in, entitlement, and PIN gates ship only in the pilot build. The pilot build is distributed to testers under the author's pilot evaluation terms, not under the AGPL, which grants no rights in the pilot application. The source repository is private during the pilot, so the clone URL and contribution instructions below describe the repository as it will be published; references to "this repository", `LICENSE`, and `COMMERCIAL-LICENSE.md` mean files in that repository. Access is available on request from the author; licensing inquiries go to the author as well.

Companion documents in this report. A validation note summarizing the March 2026 cross-model comparison, the July 2026 comparison with human-assigned reference scores, and the

June 2026 test-retest stability study (all run on development builds before the 0.1.2 pilot build and not re-run, so they do not measure the effect of the 0.1.10 changes), plus the architecture and grading-pipeline diagrams as standalone image files. The full comparison report carries item-level detail from one course section and is available from the author on request. The current public description of the project is maintained at <https://travisfraser.com/rubrica>.

Rubrica grades handwritten student exams locally. Under the default local-Ollama configuration, cover-page extraction runs on the instructor's machine. Stored names, SIDs, and page 0 are excluded from external grading requests; those requests contain an anonymous identifier, answer-page images, and rubric text. Because answer pages and rubrics can themselves contain incidental identifiers, instructors should review and remove that information before external grading. The system was built while the author served as a Graduate Student Instructor for Microeconomics at UC Berkeley Haas, and it has been used in production across multiple courses and semesters, with 600+ exams graded.

What separates Rubrica from a thin wrapper around a vision API is the *mechanism*. The grading pipeline is layered and stability-tested, with author-run cross-model comparisons as a structural check. Pre-processing applies 2x contrast enhancement to surface faint pencil; the API call uses temperature 0.0 wherever the API exposes it, which minimizes but does not eliminate run-to-run variation; sub-part consolidation, score normalization, and feedback sanitization happen in Python after the response; vague feedback is detected and refined; contradictions between feedback and assigned score are caught and resolved; scores within 1.5% of letter-grade thresholds are automatically re-graded in a second pass. The claim to reliability is grounded in measurement, not appearance.

This repository is the open-source core

Rubrica is **open-core**. This repository is the AGPL-licensed core: a complete, working grader, not a crippled demo. The commercial edition adds proprietary modules that are not published here, listed under Commercial edition. See `COMMERCIAL-LICENSE.md` and License.

HOW IT WORKS

1. **Upload a rubric.** PDF or DOCX per exam version, sent natively to Claude so tables and formatting are preserved.
2. **Upload a roster** (*optional*). CSV of student names and SIDs for fuzzy matching during name assignment.
3. **Upload exams.** Up to three batch PDFs at once; the app splits all of them by page count and merges them into one review session.
4. **Assign names.** Under the default local-Ollama configuration, a vision model reads each cover page on the instructor's machine to pre-fill student name and SID; fuzzy roster matching corrects OCR errors; extraction runs in the background with live progress and an abort button.

5. **Grade.** Claude Sonnet receives an anonymous ID, answer-page images, and rubric text. Stored names, SIDs, and page 0 are excluded, but answer pages and rubrics may contain incidental identifiers. Boundary scores are re-graded in a second pass; vague feedback is refined to cite specific rubric criteria; multiple-choice zeros trigger a focused re-read; feedback that contradicts its own score is caught and resolved.
6. **Review and adjust.** Inline score editing per question or sub-part, per-student printable reports, summary and detailed CSV exports. All grades are mapped back to names locally, with a full edit-log audit trail.
7. **Save and switch.** Save the entire grading session (database, uploaded files, rubric) as a named, versioned snapshot, and load any prior snapshot back into the active workspace.

The commercial edition extends this loop with AI rubric enhancement, an in-app assistant, an author-run cross-model comparison, remote GSI review, analytics, and email distribution. See Commercial edition.

FEATURES (OPEN-SOURCE CORE)

Upload and review

- **Multi-batch upload:** up to three batch PDFs at once, split by page count and merged into one review session
- **Roster upload:** CSV import of student names and SIDs for fuzzy matching during name assignment
- **Cover-page vision:** under the default local-Ollama configuration, a vision model auto-extracts names and SIDs on the instructor's machine, abortable with live progress
- **Cover-page consistency check:** a perceptual hash flags exams whose cover page differs from the majority
- **Roster fuzzy matching:** local name and SID matching with character-confusion corrections (0 to 0, 1 to 1)
- **Scan modal:** page-by-page exam viewer for verifying all pages are attached before confirming
- **Persistent review tab:** a nav tab appears whenever an unsaved review session exists; it survives page navigation and restarts

Grading accuracy

- **Boundary re-grading:** scores within 1.5% of letter-grade thresholds (90/80/70/60) trigger a fresh second pass; disagreements are averaged with a full decision trail
- **Feedback specificity enforcement:** vague feedback ("Good work", "Incorrect", under 8 words) is refined via a follow-up API call requiring rubric-anchored justification
- **Contradiction handling:** feedback whose substance disagrees with the assigned score (praise paired with a deduction, criticism paired with full marks) is detected and the affected items are re-graded

- **MC double-read verification:** any multiple-choice question scored zero triggers a focused second API call that re-reads only the relevant page and identifies the circled letter; disagreements are flagged for instructor review
- **Handwriting confidence flag:** the grading model reports a per-question handwriting confidence; ambiguous or partially illegible items are surfaced for verification before the exam can be marked reviewed
- **Crossed-out answer detection:** the grading prompt explicitly ignores answers struck through with slashes or Xs; only cleanly circled responses are scored
- **Prompt-injection guard:** the grading prompt treats any instruction written on an exam page as student content, never as a directive; suspected injection attempts are flagged for human review
- **Pre-flight cost estimate:** every grading confirmation shows the exam count and an estimated dollar range derived from the workspace's own usage history; nothing is charged without that confirmation
- **Faint pencil handling:** 2x contrast enhancement applied to all exam pages before grading
- **Low-variance scoring:** temperature 0.0 wherever the API exposes it (greedy decoding, which minimizes but does not eliminate run-to-run variation); models without sampling controls (Opus 4.7+, o-series) run at the vendor default. Residual variance is measured by a test-retest stability audit rather than assumed away
- **Feedback sanitizer:** strips AI deliberation language, scoring verdicts, inline point tallies, and em or en dashes from all student-facing feedback
- **Sub-part consolidation:** multi-part questions (Q3a, Q3b, Q3c) are graded individually then consolidated into Q3 totals with concatenated feedback; the pre-consolidation breakdown stays visible for review
- **Shared scoring package:** `scoring/` (consolidation, sanitization, finalization, boundary, specificity, subparts, flags) is the single source of truth for all scoring math
- **Raw score snapshot:** scores are copied to a `raw_scores` field after the shared sub-part consolidation and feedback sanitization steps and before the production-only safeguards (feedback refinement, contradiction resolution, boundary re-grade), so the cross-model comparison sees the same processing stage in both pipelines
- **JSON retry:** automatic retry on malformed model responses with persistent error logging
- **Review-flag safeguard:** exams with unresolved handwriting, MC-mismatch, or contradiction flags cannot be marked reviewed until an instructor dismisses each one

Privacy

- **External grading request boundary:** cryptographically random identifiers (`secrets.token_urlsafe`) replace stored student names and SIDs; requests contain the identifier, answer-page images, and rubric text. Answer pages and rubrics may still contain incidental identifiers
- **Cover-page exclusion:** page 0 is not sent to the external grading provider. Under the default local-Ollama configuration, cover OCR runs on the instructor's machine

- **Private Mode:** a single toggle hides all names and SIDs and blurs cover pages across every page, designed for screen-sharing
- **Local-first storage:** working data (database, PDFs, grades) stays on the instructor's machine unless the instructor turns on the optional backup (off by default)
- **Local roster matching:** fuzzy name and SID matching runs entirely in Python, with no external call

Results and reports

- **Inline score adjustment:** edit per-question and per-sub-part earned points after grading; saves via AJAX with live preview
- **Edit-log audit trail:** every grade change is recorded with timestamp, actor, anon_id, source, field, old value, and new value
- **Reviewed checkmark:** a per-exam toggle to track review progress, persisted across sessions with row highlighting
- **Sub-part breakdown:** a toggleable panel showing pre-consolidation sub-question scores with sanitized feedback; never shown on student reports
- **Grade management:** grade, re-grade, or clear grades per-exam or in bulk; Select Ungraded for batch operations
- **Per-student PDF reports:** a printable per-student report with score card, progress bar, question breakdown, and separated sub-part feedback
- **CSV export:** summary (one row per student), detailed (one row per question), and sub-parts (one row per sub-question) formats
- **Grading overview report:** a printable per-version overview of the graded batch

Workspaces and snapshots

- **Workspace switching:** save the entire grading session (database tables, uploaded files, rubric) as a named snapshot, and load any prior snapshot back into the active workspace
- **Versioned snapshots:** each save lands in a fresh `v<utc_ts>/` subdir with a `latest` text-file pointer; retention defaults to 5 versions, configurable; the active version is never pruned
- **Atomic writes:** saves go to `<vid>.tmp/`, fsync, then `os.replace` to `<vid>/`; the pointer only advances after the rename, so an interrupted save leaves the prior version authoritative
- **Optional off-disk backup (off by default):** setting `RUBRICA_BACKUP_DIR` turns it on and names the directory; every successful save then kicks off an async background copy to that directory; if it is cloud-synced, the snapshot, including the local name mapping, follows that service's terms
- **Auto-save on exit:** clean shutdown flushes the active workspace to a new version dir, gated so it never fires mid-grading or on a deliberately empty workspace

Infrastructure

- **Parallel grading:** five concurrent workers with WAL-mode SQLite for safe concurrent writes

- **Backend-agnostic data layer:** the v0.1.10 build uses SQLite in WAL mode; the `db.py` accessors are designed to support a later Postgres cutover without changing call sites
- **Live progress:** background grading with a progress bar that persists across page navigation
- **API resilience:** a 5-minute request timeout and 10-second connect timeout prevent worker deadlock; malformed responses are retried before erroring
- **Rubric versioning:** multiple rubric versions (A, B, ...) for different exam variants
- **Batch resume:** Grade All skips already-graded exams, so re-runs never duplicate API calls
- **Localhost guard:** the app rejects non-local Host headers and cross-origin POSTs (CSRF and DNS-rebinding protection for a deliberately local server)
- **Prompt caching** (*on by default; `RUBRICA_PROMPT_CACHE=0` turns it off*): caches the shared system-plus-rubric prefix across exams of a version (roughly 40% off grading input cost). A June 2026 check of a release candidate that included caching (alongside two other changes) found scores within ordinary run-to-run variation, not identical outputs
- **Input validation:** session IDs validated on all routes; error messages sanitized
- **Upload size limit:** a 50 MB cap per batch PDF
- **Dark mode** and **built-in docs** at `/docs`, printable as PDF
- **Open-core leak guard:** `scripts/check_public_safe.py` (run in CI and pre-commit) enforces that no proprietary module, secret, or unlisted file can enter the core repository

COMMERCIAL EDITION

The commercial edition adds proprietary modules that are not part of this repository. A commercial license also permits use of the core without the AGPL's network-copyleft obligations. To inquire, see `COMMERCIAL-LICENSE.md`.

- **AI rubric intelligence:** extracts questions, max points, and answer keys from an uploaded rubric into structured form, with interactive refinement and cross-version mapping
- **Author-run cross-model comparison:** re-grades stratified samples with a cross-family reference scorer and produces a faculty-ready PDF report with agreement metrics (exact match, QWK, ICC, MAE, observed mean difference). The methodology and representative results are documented under Validation and research
- **In-app assistant (Ask Rubi):** a stage-aware assistant that answers "how do I" questions, explains review flags, and proposes (never applies) score or rubric changes, behind a PII wall that keeps names away from the model
- **Remote GSI review:** an isolated, read-only, token-gated review server so graders can propose changesets that the head grader approves before they land in the canonical database
- **Analytics dashboard:** grade-distribution and letter-grade charts and per-question difficulty analysis with version filtering
- **Email distribution:** per-student PDF report delivery via Gmail OAuth2 with per-recipient tracking
- **Advanced reports:** additional faculty-facing diagnostic and presentation reports

TECH STACK

LAYER	TECHNOLOGY
Web framework	Flask 3.x
AI grading	claude-sonnet-4-6 (vision, via the Anthropic API)
AI name extraction	qwen2.5v1:3b via Ollama (local, on-device)
PDF handling	pypdf, pdfplumber, pypdfium2, reportlab
Document parsing	python-docx
Database	SQLite (WAL mode) via a backend-agnostic shim
Frontend	Bootstrap 5, Chart.js 4, Jinja2

SETUP

Requirements: Python 3.10+, an [Anthropic API key](#), and [Ollama](#) for local cover-page OCR (optional; without it, enter names manually).

Clone the repository (private during the pilot; the URL resolves once it opens):

```
git clone https://github.com/travisstephenfraser/Rubrica.git
cd Rubrica
```

Contributing. Activate the repo's git hooks once with `bash scripts/install_hooks.sh` (points git at `scripts/git-hooks/` via `core.hooksPath`). The pre-commit and pre-push hooks run `scripts/check_public_safe.py` and `scripts/check_licenses.py` so no secret, PII, or copyleft dependency can be committed or pushed. The same checks run in CI on every push and pull request. Contributions are accepted only under a contributor license agreement that permits the core to be offered under the commercial license as well as the AGPL.

Quick start (one command). After cloning, run `./run.sh` (macOS/Linux) or `run.bat` (Windows). It creates the virtual environment, installs dependencies, scaffolds a `.env` for your API key, and opens the app in your browser. Re-run it any time. Install [Ollama](#) separately if you want local cover-page OCR (otherwise enter names manually). The per-platform steps below are the manual equivalent.

Try it at zero cost. `python scripts/load_demo.py` loads three synthetic pre-graded exams with no API key and no API spend, showing the happy path, the review queue with every flag type, and a grade-boundary case. It creates its own demo PDFs and database on a fresh clone, so nothing else needs to run first. `python scripts/load_demo.py --remove` cleans them up.

macOS

Tested on Apple Silicon and Intel Macs running macOS 13 Ventura or later. Ollama uses the Apple Metal GPU backend automatically.

Install Python and Ollama with [Homebrew](#):

```
brew install python@3.12 ollama
```

Create a virtual environment and install dependencies (a venv is recommended because Homebrew/system Python is an externally managed environment):

```
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
```

Start Ollama and pull the local vision model (one-time, ~3.2 GB):

```
brew services start ollama      # or run `ollama serve` in a separate terminal
ollama pull qwen2.5vl:3b
```

Create a `.env` file (see Configuration), then run:

```
python grader.py
```

Open `http://localhost:5000`.

macOS port 5000 conflict: macOS Monterey and later run an AirPlay Receiver that binds `*:5000` and returns `403 Forbidden`. If you hit a 403, either turn off **System Settings, General, AirDrop & Handoff, AirPlay Receiver**, or run on another port with `RUBRICA_PORT=5050 python grader.py`.

Windows

```
pip install -r requirements.txt
ollama pull qwen2.5vl:3b      :: one-time, ~3.2 GB
```

Create a `.env` file, then run (the encoding flag is required for emoji output):

```
set PYTHONIOENCODING=utf-8
python grader.py
```

Open `http://localhost:5000`.

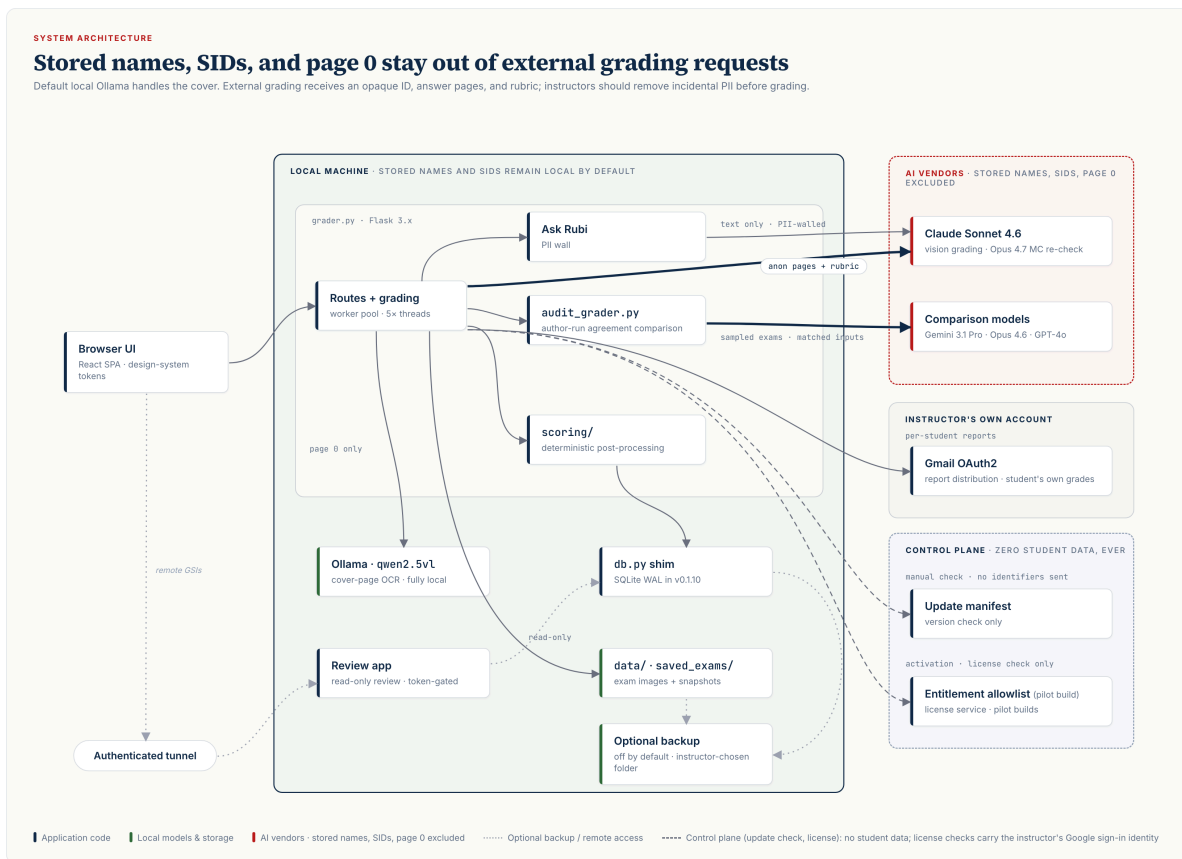
Configuration

Create a `.env` file in the project root:

```
ANTHROPIC_API_KEY=your_key_here
OLLAMA_EXE=C:\path\to\ollama.exe # optional, Windows only, if ollama is not in PATH
RUBRICA_PORT=5000 # optional; use 5050 on macOS if AirPlay holds 5000
RUBRICA_BACKUP_DIR=/path/to/backup # optional; backup folder for workspace saves; setting it turns backup on
```

ARCHITECTURE

The file tree below shows *where code lives*; this section shows *how the system runs*. Under the default local-Ollama configuration, stored names, SIDs, and page 0 remain on the local machine while external grading receives an opaque `anon_id`, answer-page images, and rubric text. Answer pages and rubrics may still contain incidental identifiers. The diagram is organized around that boundary rather than around the module layout.



Rubrica architecture: the privacy boundary between the local machine and external APIs

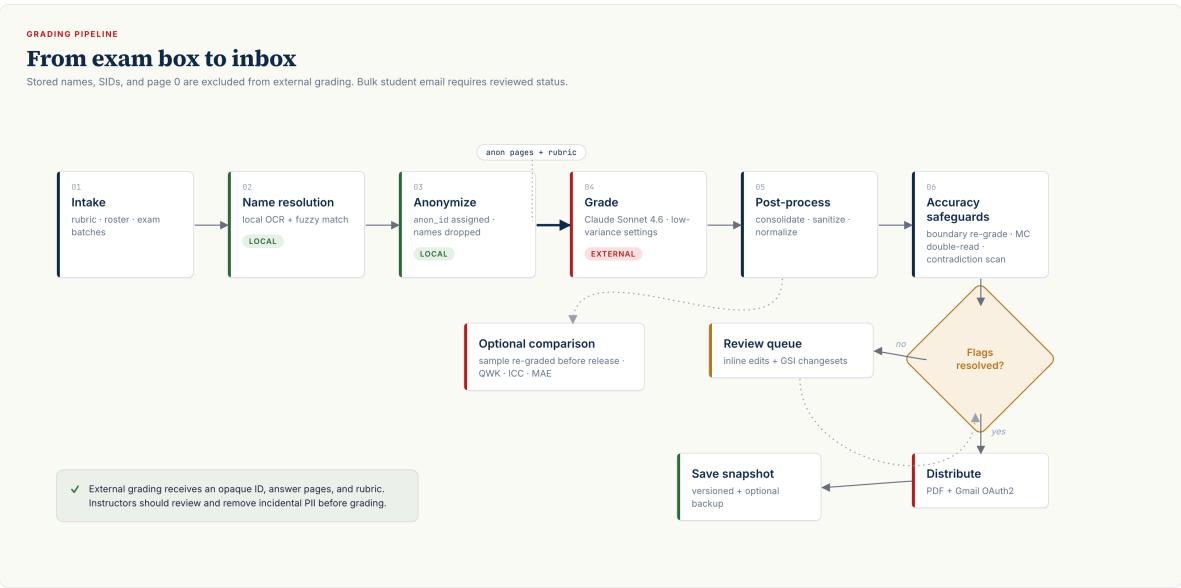
Thick arrows (`=>`) are calls that cross the privacy boundary. Under the default local-Ollama configuration, grading requests omit stored names, SIDs, and page 0. Answer pages and rubrics can

contain incidental identifiers unless the instructor reviews and removes them before external grading.

Both diagrams are also included in this record as standalone PNG files.

Grading process flow

The diagram above is the static component view; this one is the dynamic view — a single exam’s path from upload to delivered grade. Red nodes are external API calls, the yellow diamond is the review gate, green nodes anonymize or persist data. The “Accuracy safeguards” stage expands into seven distinct checks; each of those checks is listed under Grading accuracy above.



Grading pipeline for a single exam, from exam box to inbox

Building blocks and technologies

BLOCK	TECHNOLOGY	RESPONSIBILITY
Web UI	Bootstrap 5, Chart.js 4, Jinja2	Upload, review, inline grade editing, analytics, all server-rendered
Main app	Flask 3.x (grader.py)	Routes, request handling, the grading worker pool, workspace save/load
Grading worker pool	ThreadPoolExecutor , 5 workers	Fans exam grading out across concurrent API calls

BLOCK	TECHNOLOGY	RESPONSIBILITY
Scoring pipeline	Pure Python (<code>scoring/</code>)	Deterministic post-processing: sub-part consolidation, sanitization, normalization, boundary reconciliation, specificity refinement
Grading model	<code>claude-sonnet-4-6</code> (vision, Anthropic API)	Reads answer pages, scores against the rubric at temperature 0.0
Cover-page OCR	<code>qwen2.5vl:3b</code> via Ollama (local)	Extracts names/SIDs from page 0 on-device; never calls out
Comparison pipeline	<code>audit_grader.py</code> + Gemini 3.1 Pro (default) / Opus 4.6 / GPT-4o	Author-run cross-model re-grading of stratified samples for agreement metrics
Ask Rubi	<code>rubi/</code> + Anthropic API	In-app assistant behind a PII wall (anon context only)
Data layer	<code>db.py</code> shim over SQLite (WAL)	Backend-agnostic accessor in preparation for a later Postgres cutover; v0.1.10 is SQLite-only
Storage	Filesystem (<code>data/</code> , <code>saved_exams/</code>)	Anonymized PDFs, rubrics, versioned workspace snapshots, logs
GSI review	Separate read-only review app, reachable by remote GSIs only through an authenticated outbound tunnel	Isolated, token-gated remote review with changeset approval
Distribution	<code>gmail_sender.py</code> (Gmail OAuth2)	Per-student PDF report delivery with per-recipient tracking

Bottlenecks

- **The grading API call is the throughput ceiling.** Each exam is a network round-trip to Claude with vision input, and boundary cases trigger a second pass. This is mitigated three ways: 5 parallel workers, prompt caching of the shared system+rubric prefix (on by default, `RUBRICA_PROMPT_CACHE=0` turns it off; roughly 40% off input cost), and batch resume so re-runs never re-grade an already-graded exam. The practical limit beyond that is the Anthropic account's own rate limit, not the app.
- **SQLite is single-writer.** WAL mode allows concurrent reads alongside one writer, which is sufficient for one instructor driving five workers. The v0.1.10 build does not support multiple machines writing to one database.

- **Cover-page OCR is GPU-bound and local by design.** It runs in a background thread with live progress and an abort button, so it never blocks the grading path; it is slow on CPU-only hardware but that is an accepted cost of keeping names on-device.
- **Save / load / clear are serialized** behind a single mutex (`_save_load_lock`); a snapshot operation in flight makes the next one fail fast rather than queue, which is deliberate.

Scalability, redundancy, and availability

- **The data layer is the planned scale path.** Version 0.1.10 runs on SQLite on one machine. A Postgres implementation exists on a separate internal branch but is not active in this build; the `db.py` shim was designed so a later cutover does not require rewriting call sites. Grading throughput can still be adjusted through the worker count within the API rate limit.
- **Redundancy is layered.** Workspace saves are atomic and versioned (`<slug>/v<utc_ts>/` , write to `.tmp` then `os.replace` , keep the last 5); when the instructor sets `RUBRICA_BACKUP_DIR` , every save is also copied asynchronously to that directory; the author's own deployment adds an operator-scheduled nightly backup of the database (newest 14 point-in-time copies); it is not part of the open core or the pilot app. With backup off (the default), the saved versions and the at-exit autosave sit on the same disk, and the autosave cannot fire on hard-kill or power loss; the optional backup directory is an extra layer.
- **Availability favors failing safe over failing open.** API calls carry a 5-minute request and 10-second connect timeout to prevent worker deadlock; malformed model responses are retried (max 2) before erroring; the GSI server is read-only and approval-gated by construction, so a compromise there cannot write to the canonical DB. When a privacy question is ambiguous, the system declines rather than proceeds.

PROJECT STRUCTURE

```
Rubrica/
├── grader.py           # Flask app: routes, grading worker pool, init_db
├── db.py              # Backend-agnostic DB accessor
├── rendering.py       # PDF page rendering to images for vision input
├──
├── scoring/          # Single-source-of-truth scoring pipeline
│   ├── consolidation.py # Sub-part to Q-level rollup
│   ├── sanitization.py  # Feedback cleaner (deliberation, dashes)
│   ├── finalize.py      # Score recalculation, normalization, letter grade
│   ├── boundary.py      # Boundary re-grade reconciliation
│   ├── specificity.py    # Vague-feedback detection and refinement
│   ├── subparts.py      # Sub-part derivation and concatenation
│   ├── flags.py         # Anomaly flag aggregation
│   └── _jsonparse.py    # Tolerant JSON extraction from model output
├──
├── reports/
│   └── generate_grading_overview.py
├──
```

```

├── scripts/
│   ├── load_demo.py           # Zero-cost synthetic demo workspace (no API key)
│   ├── check_public_safe.py   # Open-core leak guard (CI + pre-commit)
│   ├── check_licenses.py      # Copyleft-dependency guard
│   └── pre-commit              # Runs the guards before each commit
├── templates/                 # Jinja2 + Bootstrap 5 templates
├── .github/workflows/guard.yml # CI: leak + copyleft guards on push/PR
├── run.sh / run.bat           # One-command launcher (macOS-Linux / Windows)
├── .env.example               # Config template (copy to .env)
├── requirements.txt
├── public_manifest.txt        # The allowlist defining this public surface
├── LICENSE                    # AGPL-3.0
├── COMMERCIAL-LICENSE.md
└── THIRD_PARTY_LICENSES.txt

```

Student data (`data/` , `saved_exams/`) is created at runtime and is gitignored; it never enters the repository.

VALIDATION AND RESEARCH

Rubrica uses an author-run, matched-input cross-model comparison as a structural check. A stratified sample is re-graded with matched rubric content and rendered answer pages, then processed through shared Python scoring functions. Provider-specific message roles, request formatting, and inference controls differ. The comparison implementation also uses production output to supply `max_points` and missing-question structure. The results therefore estimate observed cross-model concordance; they are not a third-party audit or an experiment in which the model is literally the only variable. Using a model from another vendor reduces, but does not remove, the risk of correlated errors.

Representative author-run cross-model comparison (n=30, 840 scored items, run March 2026)

METRIC	RESULT	BENCHMARK
Exact score match	88.3% (742 of 840)	task-dependent; no matched human-human comparison exists for this item mix
Within-1-point agreement	96.2%	internal operating criterion: at least 95%
Mean absolute error	0.15 pt per question	no universal threshold; scale-dependent
Mean difference	+0.64 pt per exam (production minus reference)	observed sample result; uncertainty spans zero

METRIC	RESULT	BENCHMARK
Quadratic Weighted Kappa (letter grades)	0.883	0.70 is a historical e-rater benchmark, not a universal standard
ICC(3,1) consistency	0.966 (95% CI 0.93 to 0.98)	descriptive only; see the correction note below

The observed production-minus-reference sample mean difference was +0.64 points per exam. In this sample the standard deviation was about 3.00 points, and an approximate 95% confidence interval for the mean runs from -0.49 to +1.76, so the direction of the population mean difference remains uncertain. Every letter-grade mismatch was between adjacent letters. Seven of nine production scores were inside the +/-1.5% boundary window; two were outside it. The comparison measures pre-safeguard scores, so post-safeguard agreement with the reference model remains unmeasured.

Correction (September 2026): earlier versions of this document reported ICC 0.964. That value came from an implementation that left the between-rater variance in the residual term; recomputed from the same data with the standard two-way formula, consistency ICC(3,1) is 0.966 (95% CI 0.93 to 0.98). ICC bands (Koo & Li, 2016) are descriptive and should be read with the confidence interval, not as an exam-grading certification threshold.

The comparison reports agreement measures used in automated-scoring research, following the evaluation framework in Williamson et al. (2012). Gradescope was the interface in which the human reference scores for Study 2 were recorded; it is not an independent auditor in this study.

Research-backed safeguards (in the open core)

- **Boundary re-grading** addresses the documented concentration of inter-rater disagreements at letter-grade thresholds. Exams within 1.5% of cutoffs (90/80/70/60) are re-graded and averaged on disagreement, with a full audit trail.
- **Feedback specificity enforcement** asks for feedback that points to the student's own work. Feedback that is very short or generic is detected and re-requested through a targeted follow-up prompt. The rule is motivated by research on how students perceive AI and human feedback (Nazaretsky et al., 2026); that study does not evaluate this rule, and Rubrica has not measured its effect on feedback quality.
- **Contradiction handling** addresses the failure mode where assigned scores diverge from the substance of the model's own feedback; affected items are re-graded.
- **MC double-read verification** addresses letter-reading errors on multiple-choice items, which carry a per-question swing. Each MC zero triggers a focused second read of the relevant page.
- **2x contrast enhancement** increases contrast on every page to help make faint pencil legible to the model. Rubrica has not measured whether this reduces any handwriting-related scoring bias.
- **Low-variance scoring:** temperature 0.0 wherever the API exposes it (greedy decoding, which minimizes but does not eliminate run-to-run variation); models without sampling controls (Opus

4.7+, o-series) run at the vendor default. Residual variance is measured by a test-retest stability audit rather than assumed away.

- **Score recalculation in Python** runs after the API response, so arithmetic consistency does not depend on model output.

Study design and sample size

There is no single sample size that makes a standard deviation, QWK, or ICC reliable. Precision depends on the score scale and category frequencies, the expected reliability, the number of raters or repeated runs, and the desired confidence-interval width. Each study should state its sampling design and uncertainty rather than appeal to a universal cutoff. Product thresholds such as 95% within one point are internal operating criteria, not general automated-scoring standards.

For first-time exams, human calibration is essential and complements cross-model comparison. The cross-model check can expose pipeline regressions and parsing failures; comparison with qualified human-assigned reference scores is the accuracy signal. The number of human-scored exams should be chosen for the exam, cohort, score scale, and required precision, then reported with uncertainty before grades go to students.

Study 2: Comparison with human-assigned reference scores (July 2026)

A macroeconomics midterm from another instructor, with human scores recorded in Gradescope, 25 exams joined per question. On the decomposed section (38 items, 46.5 points) Rubrica's mean absolute difference from the human score was 1.26 points per exam (2.7% of the section), with an observed mean difference of -0.28 points, and 85.4% of compared item scores matched exactly (810 of 949; one missing model score is excluded). The one holistically scored 25-point essay had a mean absolute difference of 3.88 points and a mean difference of -3.64 points. On whole-exam totals, absolute-agreement ICC(A,1) was 0.868 (95% CI 0.42 to 0.96) and consistency ICC(C,1) 0.922 (95% CI 0.83 to 0.96); the wide lower bound reflects the essay's systematic difference. The decomposed section disagreed less than the essay, but item format, point scale, and content also differ, so this is consistent with the idea that narrower criteria help, not causal evidence for it. One exam from one course: a sample-bounded comparison, not a general error rate.

Study 3: Test-retest stability (June 2026)

Ten final-exam submissions were selected by deterministic stratification across exam version, batch, and score range, then graded three times through the production pipeline on their native rubric point scale. All 10 retained the same letter grade across the three runs, and the mean within-exam standard deviation of total score was 0.91 points, computed as the population standard deviation of each exam's three totals (dividing by 3); the sample standard deviation (dividing by 2) averages 1.11 points. The noisiest item was SA1. A separate earlier 30-exam, three-run capture observed 1.03 points; which convention that figure used was not recorded, so it is context only and not directly comparable to either value above.

This is a small test-retest study from one workspace with three repeats per exam. It estimates variation for that setting only, supplies no cross-course reliability estimate, and does not test accuracy against a human. Student-level results are not deposited; the record reports aggregate results to preserve privacy.

Key references

- Williamson, D. M., Xi, X., & Breyer, F. J. (2012). A framework for evaluation and use of automated scoring. *Educational Measurement: Issues and Practice*, 31(1), 2-13. [doi:10.1111/j.1745-3992.2011.00223.x](https://doi.org/10.1111/j.1745-3992.2011.00223.x)
- Singh, A., Karayev, S., Gutowski, K., & Abbeel, P. (2017). Gradescope: A fast, flexible, and fair system for scalable assessment of handwritten work. *ACM L@S*. [doi:10.1145/3051457.3051466](https://doi.org/10.1145/3051457.3051466)
- Koo, T. K., & Li, M. Y. (2016). A guideline of selecting and reporting intraclass correlation coefficients. *Journal of Chiropractic Medicine*, 15(2), 155-163. [doi:10.1016/j.jcm.2016.02.012](https://doi.org/10.1016/j.jcm.2016.02.012)
- Bonett, D. G. (2002). Sample size requirements for estimating intraclass correlations with desired precision. *Statistics in Medicine*, 21(9), 1331-1335. [doi:10.1002/sim.1108](https://doi.org/10.1002/sim.1108)
- Landis, J. R., & Koch, G. G. (1977). The measurement of observer agreement for categorical data. *Biometrics*, 33(1), 159-174. [doi:10.2307/2529310](https://doi.org/10.2307/2529310)
- Fleiss, J. L. (1981). *Statistical Methods for Rates and Proportions* (2nd ed.). Wiley.
- Gwet, K. L. (2014). *Handbook of Inter-Rater Reliability* (4th ed.). Advanced Analytics, LLC.
- Nazaretsky, T., et al. (2026). Who gives feedback matters: student biases towards human and AI-generated formative feedback. *Journal of Computer Assisted Learning*. [doi:10.1111/jcal.70153](https://doi.org/10.1111/jcal.70153)
- Sim, J., & Wright, C. C. (2005). The kappa statistic in reliability studies. *Physical Therapy*, 85(3), 257-268. [doi:10.1093/ptj/85.3.257](https://doi.org/10.1093/ptj/85.3.257)

PRIVACY MODEL

Privacy is the architectural constraint the rest of the system is built around, not a presentation layer bolted on. The `data/` and `saved_exams/` directories are gitignored. Under the default local-Ollama configuration, cover-page OCR runs on the instructor's machine. Stored names, SIDs, and page 0 are excluded from external grading requests; those requests contain an anonymous identifier, answer-page images, and rubric text. Rubrica does not inspect answer pages or rubrics for incidental identifiers, so instructors should review and remove that information before external grading. Roster fuzzy matching runs entirely in Python with no external call. *Private Mode* hides names in the UI during screen-sharing while leaving them intact in the database and form submissions.

LICENSE

Rubrica is **open-core**. This repository's core is licensed under the GNU Affero General Public License v3.0 (AGPL-3.0) and will be published when the repository opens after the pilot: a complete, working

grader. The commercial edition adds proprietary modules (AI rubric intelligence, dual-model cross-model comparison, the in-app assistant, remote GSI review, analytics, email distribution, and advanced reports) and a license to use the core without AGPL obligations. See `COMMERCIAL-LICENSE.md` .

Under the AGPL you are free to use, modify, and distribute the core for any purpose, including academic and personal use. If you deploy a modified version as a network service, the AGPL requires you to make your source code available. To use Rubrica without those obligations, or to license the proprietary modules, contact the copyright holder, Travis Fraser.

Third-party dependency licenses and attributions are listed in `THIRD_PARTY_LICENSES.txt` .