ImgGrader: User Guide

Overview

ImgGrader is a Python GUI application for automated grading of image-processing assignments. It compares student outputs against expected references using a rubric composed of rules (metrics, tolerances, scoring mode, etc.). Rules can be added manually or imported/exported as JSON; results are saved to CSV. The app supports single and batch evaluation, region-of-interest (ROI) masks, and console-output checks.

1 Main workflow (step by step)

Below is the recommended sequence for typical use; corresponding UI elements are shown in Figures 1–4.

1. **Select the expected image, then select student images.** Use the top buttons *Load Expected Image* and *Load Student Image*. The expected image is the ground truth or reference. Student images are the outputs to be graded.

2. **Add rules (manual).** Click *Add Rule (popup)* to open the rule editor (Figure 2a). You will specify:

   - **Output (student filename):** the name (or pattern) of the student output file the rule applies to.
   - **Metric:** choose one of the listed metrics (Table 1).
   - **Aggregation:** how to combine per-pixel or per-tile values (e.g., mean/median/max/trim10).
   - **Max Score:** the points this rule contributes.
   - **Tolerances:** thresholds used by bucket/continuous scoring; syntax examples are in the UI help and Table 2.
   - **Expected console output (if using EXACT/regex/contains for `cout`):** the expected text to match.
   - **Scoring mode:** *bucket* (discrete levels) or *continuous* (interpolated).
   - **Keywords/notes:** optional key–value hints, e.g., `mask=./expected/H01_fov.png`, `gray=1`, etc.

   Rules can be removed with *Delete Rule* Button.

3. **(Alternative) Import ready JSON rules.** Click *Import Rules (JSON)* to load a previously saved rubric.

4. **Export rules (JSON).** Click *Export Rules (JSON)* to save the current rubric to a portable file.

5. **Run single evaluation.** Use *Run Single Eval* to score the currently shown student image against the expected one using the active rules.

6. **Batch evaluation (whole class).** Use *Select Expected Dir* and *Select Student Dir*, then click *Batch Evaluate (student dir)*. If a new student folder arrives, click *Refresh Students* to make it appear in the dropdown and re-run as needed.

7. **Refresh Text from Table.** This mirrors the rules currently visible in the table into the JSON text box (so you can export exactly what you see).

8. **Apply Text to Table.** This parses the JSON text box and updates the rules table *live* (useful when hand-editing JSON).

9. **Load Mask / ROI.** Click *Load Mask / ROI* to either draw a rectangular ROI or load a binary mask image (Figures 4a–c). Each rule can carry its own mask via `notes`, e.g., `mask=./expected/H01_bg_ring.png`.

10. **Clear Images.** Clears only the currently loaded images from the viewers (rules remain).

11. **Reset All.** Clears *everything* (images, rules, status).

12. **Mask indicator (top-right).** When a mask/ROI is active, the top-right shows *Mask:* with the source (file path or drawn ROI name). Drawn masks are saved locally for reuse.

13. **Scan Assignment Folder.** Scans for available assignments/submissions in the selected student directory structure.

2 Rule design: metrics & when to use them

Table 1 summarizes the built-in metrics and typical use cases. You select a metric per rule and set its *tolerances*, *max_score*, *scoring mode*, and (optionally) a *mask* to restrict evaluation to a region of interest.

Table 1: Core metrics and typical uses.

Metric	What it measures / When to use
`EXACT` (image)	Bit-exact equality (reference implementations, binary masks). Very strict; any difference fails.
`SIZE` (dimensions)	Width/height match; use as a quick sanity check of outputs.
MSE / MAE / RMSE	Pixel-wise error on linear intensity scale (denoising, simple filtering). MAE less sensitive to outliers; RMSE in original units.
PSNR	Log-scale quality vs. error (compression, denoising). Higher is better.
SSIM	Perceptual similarity (luminance/contrast/structure); good for blur/compression comparisons.
NCC	Pattern correlation, invariant to affine intensity changes; sensitive to misalignment.
IoU	Overlap of binary masks (segmentation/detection).
ΔE (CIEDE2000)	Perceptual color difference in CIELab (color transforms/white balance).
`N_DIFF_PIXELS`	Count of pixels whose absolute difference exceeds a threshold; useful inside special ROIs (e.g., centerline ring or background ring).
`Objects/Holes`	Miscounts vs. reference in binary masks; detects structural errors.
`Contours` / `ContoursStrict`	Boundary fidelity and topology checks; `Strict` enforces hole hierarchy.
`cout` (text)	Console-output check: exact/contains/regex match against expected text.

Aggregation Choose *mean*, *median*, *max*, or *trim10* (trimmed mean) when the metric is computed over a map or multiple tiles/regions.

Scoring *Bucket* mode maps thresholds to discrete scores; *continuous* interpolates between thresholds. Examples are shown in Table 2.

Table 2: Tolerance examples (interpreted by the rule engine).

Example	Interpretation
`0.82,0.90,0.95`	IoU buckets (e.g., 0, 1.67, 3.33, 5.0 points).
`thr:10 | 5000,30000`	For `N_DIFF_PIXELS`, threshold of 10 gray levels; buckets at 5k and 30k pixels.
`thr:10 | 0,5000`	For `N_DIFF_PIXELS`, reward few differences (e.g., background ring).

3 Masks and ROIs

Every rule can carry its *own* mask via the `notes` field (`mask=...`), making grading interpretable and robust:

- **FOV mask** for global metrics (e.g., IoU within field of view).

- **CL-ring** (centerline ring) to penalize missed/fragmented vessels.
- **BG-ring** (background ring) to penalize spurious background speckle.

You can also draw a quick rectangular ROI (*Load Mask / ROI* $\rightarrow$ *Yes*) for ad-hoc checks.

4 Buttons and utilities (summary)

- **Clear Images** removes the current expected/student previews but keeps rules intact.
- **Reset All** clears rules, images, and state.
- **Import/Export Rules (JSON)** for reusability and sharing.
- **Run Single Eval** applies rules to the current pair.
- **Batch Evaluate** traverses the student directory and scores all submissions; exports CSV.
- **Refresh Students** re-scans the student directory (handy after adding a new folder).
- **Refresh Text from Table / Apply Text to Table** keeps the UI table and JSON text synchronized.
- **Scan Assignment Folder** quickly lists available submissions.

Figures

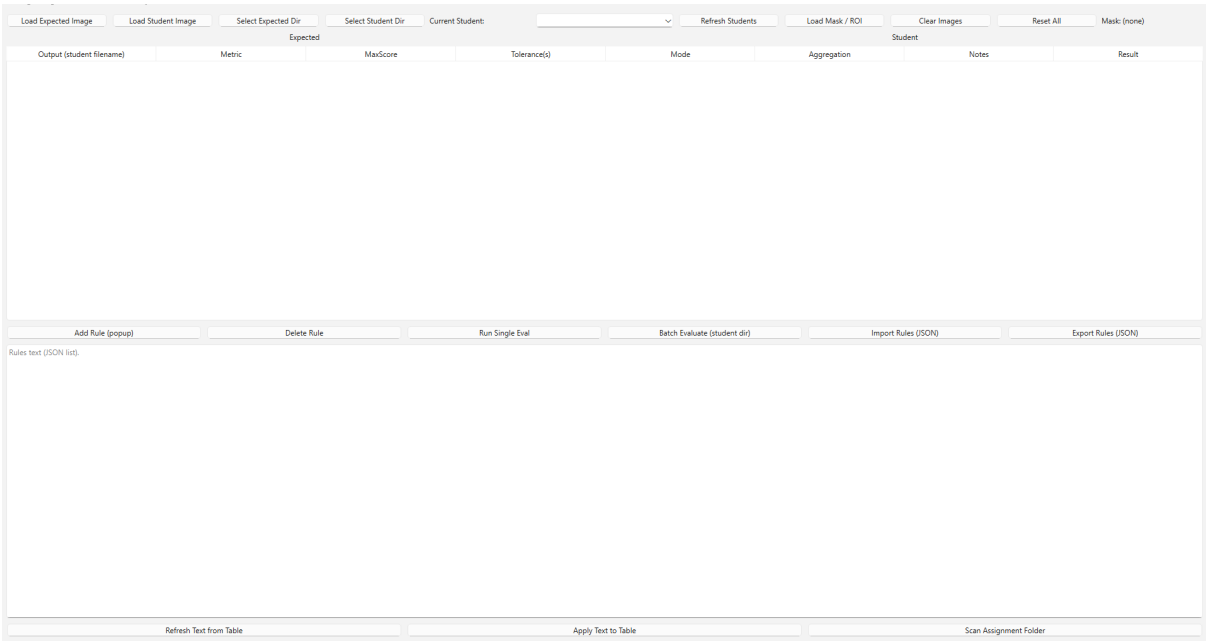
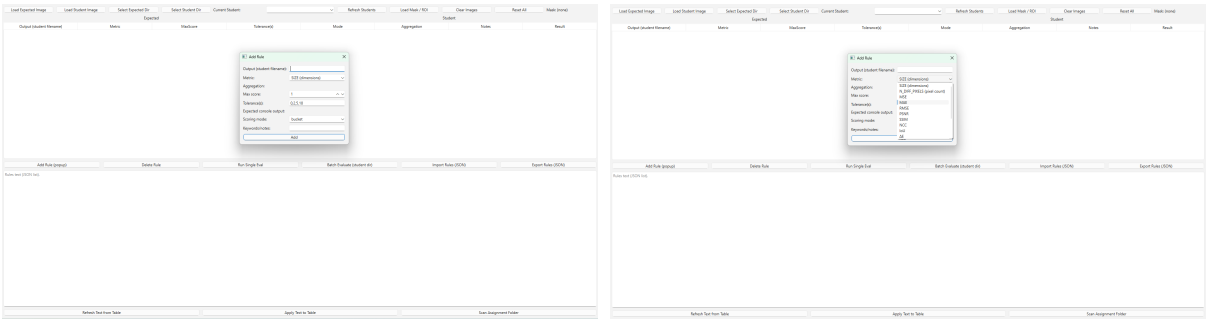


Figure 1: Main UI at startup. Top toolbar: loading expected/student images, mask/ROI, clearing and resetting, and student selector. Center: image viewers. Bottom: rules table, JSON text, and actions.



(a) Add Rule popup (manual). (b) Metric selector (scrollable list).

Figure 2: Creating rules manually.

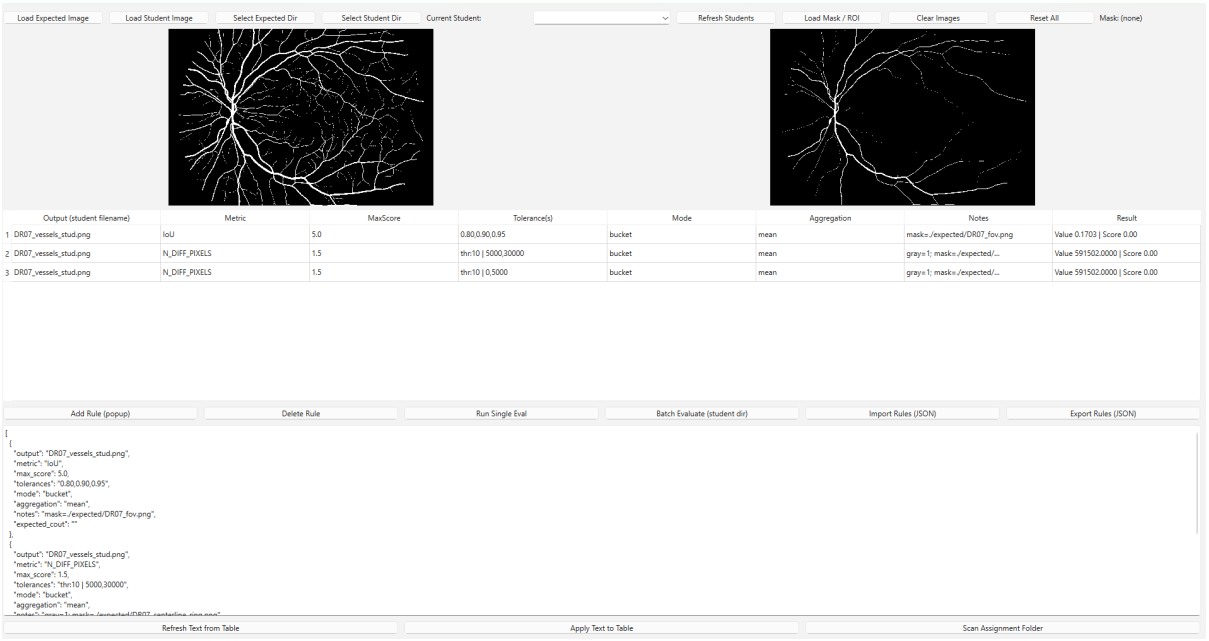
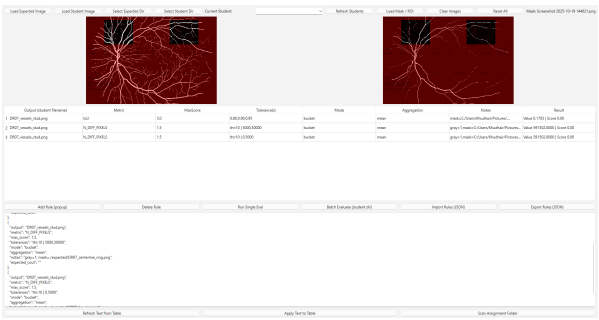
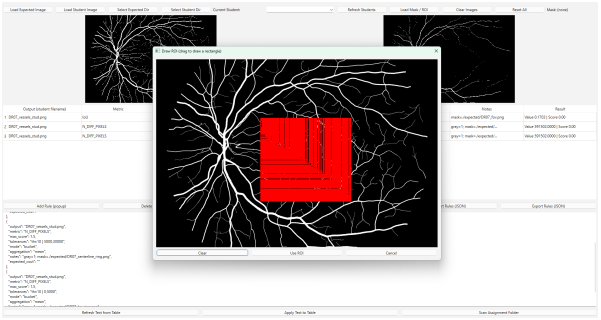


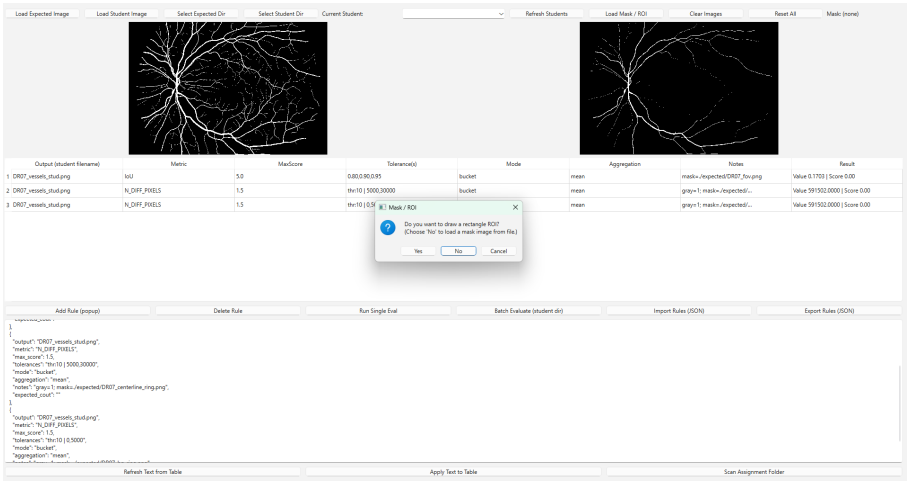
Figure 3: Live JSON view/edit. Use *Refresh Text from Table* to export visible rules into JSON, and *Apply Text to Table* to push edited JSON back to the rules table.



(a) Applying a global mask (FOV).



(b) Drawing an ROI rectangle.



(c) Selecting a mask from file.

Figure 4: Mask/ROI workflows. The top-right indicator shows the active mask name/path; drawn masks are saved locally.

Output files

- **Rule files:** JSON (human-readable, versionable).
- **Results:** CSV per batch with raw metric values, per-rule scores, and totals per student.
- **Masks:** Saved locally when drawn; external masks referenced by path in rule `notes`.