International Journal of Computer Applications Technology and Research
Volume 15–Issue 09, 01 – 10, 2026, ISSN: -2319–8656
DOI:10.7753/IJCATR1509.1001

Design and Reproducible Evaluation of a Lightweight Local Platform for Structured Image-Record Review

Zhendong Wang
College of Robotics, Beijing Union University
No. 97 Beisihuan East Road, Chaoyang District
Beijing 100101, China

Abstract: Small research teams may need to review structured image records without transferring operational files to a third-party service. This study redesigns and reproducibly evaluates a lightweight Flask platform backed by local JSON files. An isolated copy of a legacy prototype was hardened with hashed passwords, controlled registration, CSRF and role checks, structured validation, atomic replacement, validated restore, safer export and paths, login throttling, security headers, and failure-closed parsing. Evaluation used generated metadata and no operational images or accounts. All 12 functional tests passed, including an injected atomic-replacement failure that preserved the previous repository. The hardened copy satisfied all 15 audit-derived controls, versus none in the baseline; this trace is not a security certification. Across three sequential sessions, all 360 clean-start, in-process concurrency scenarios passed, allocating 32,400 synthetic records with zero duplicate identifiers. At 20 clients, median session-level P95 was 760.2 ms for batch allocation and 157.1 ms for saving one record. A separate 120-scenario loopback-HTTP run produced request-level P95 values of 913.8 and 202.7 ms. Three scale sessions yielded 900 observations from 100 to 10,000 records. A negative boundary experiment then started two or four independent processes against the same generated repository. None of 20 clean starts met the integrity criterion; 18 produced duplicate identifiers and assignment-to-master ownership mismatches, and nine also contained a write error. The evidence supports only the tested single-host, single-process configuration and empirically rejects treating its process-local lock as a multi-process transaction. Usability, review quality, production-network performance, certification, and legal compliance remain untested.

Keywords: image-record review; local web system; task allocation; data integrity; access control; reproducible evaluation

1. INTRODUCTION

Supervised vision and vision–language research depends on records whose images, text, regions, and semantic labels remain mutually consistent. Annotation software is therefore not merely a drawing interface: it determines which records are shown, what edits are permitted, how concurrent contributions are separated, and how the resulting evidence is preserved. LabelMe demonstrated an early web-based approach for collecting and sharing object annotations [1]. VIA later showed that a compact browser-based application can support image, audio, and video annotation [2]. Current general-purpose systems such as CVAT and Label Studio provide broad task coverage and extensibility [3], [4], while OneLabeler studies reusable modules for constructing varied labeling workflows [5].

The availability of capable general-purpose tools does not eliminate every task-specific implementation. The workflow examined here treats one record as a bundle containing an image reference, editable text, a keep/discard judgment, zero or more normalized regions, and two controlled label lists. Reviewers request batches, edit these fields together, and then merge accepted records into a reviewed collection. The intended setting is a small local deployment. The system should therefore operate without an external review service, while the research package and performance experiments should contain neither operational images nor real accounts. Keeping operational data local reduces unnecessary transfer, but it is not a formal privacy-preserving guarantee.

The original laboratory prototype implemented the domain workflow but had not been developed as a publishable or security-reviewed system. A source audit identified coupled integrity and web-application risks, including plaintext password comparison, source-coded registration phrases, filenames derived from usernames, state-changing GET routes, missing cross-site request forgery (CSRF) checks, validation code that was not called before persistence, non-atomic JSON writes, incomplete locking, and no validated restore operation. Rather than changing the operational copy or accessing its real data, we created an isolated research copy and evaluated it exclusively with generated metadata.

This paper asks four engineering questions:

1. **RQ1—Functional integrity and deployment boundary:** Does the research copy correctly implement authentication, role boundaries, non-overlapping task allocation, structured-record round trips, merge, export, backup, and restore under the tested single-process design, and does an independent-process falsification test expose the predicted boundary of the process-local lock? 2. **RQ2—Remediation traceability:** Are the specific weaknesses found during the baseline audit addressed by identifiable code paths and executable checks? 3. **RQ3—Concurrent local performance:** What latency and allocation integrity are observed for 1–20 concurrent clients under a fixed 1,000-record input in three in-process sessions and one loopback-HTTP session on one workstation? 4. **RQ4—Dataset-size sensitivity:** How do common JSON-backed operations behave as the generated record count increases from 100 to 10,000?

The contribution is a requirements-to-evidence case study for a bounded local workflow: identified baseline failures are linked to specific controls and executable checks; concurrency is evaluated through repeated clean starts at a controlled input size, including an actual loopback HTTP path; the storage boundary is measured separately over four dataset sizes; and the stated single-process restriction is subjected to a negative, independent-process falsification test. We do not propose a new annotation algorithm, access-control model, password scheme, or security-testing method. We also do not claim that this narrow platform replaces mature general-purpose annotation systems.

2. RELATED WORK AND DESIGN CONTEXT

2.1 Annotation systems and configurable workflows

LabelMe combined an online interface with a growing shared dataset and established a durable pattern for browser-based image annotation [1]. VIA emphasized a lightweight, offline-capable implementation and export to plain-text formats [2]. CVAT offers a general computer-vision annotation environment [3], while Label Studio supports configurable labeling over multiple data types and a modular application architecture [4]. OneLabeler abstracts common states and modules and uses visual programming to construct task-specific tools [5]. Recent systems also illustrate different forms of specialization. Annotate-Lab focuses on a simple image-annotation workflow [15], interactive pose labeling combines model proposals with human correction [16], and an astronomy toolkit couples large-image visualization with mask editing [17]. Web platforms for text annotation, including brat and INCEpTION, further show how task-specific interfaces can incorporate machine assistance without removing human control [18], [19]. These systems demonstrate two complementary directions: feature-rich general platforms and smaller tools adapted to a defined workflow.

Our work occupies a narrower point in this design space. It preserves an existing laboratory interface and record schema, packages a fixed workflow for simultaneous review of text, regions, and controlled labels, and evaluates its storage and assignment behavior. This constraint reduces installation and migration effort for the local workflow, but sacrifices the scale, integration ecosystem, and mature project administration of established platforms. The resulting system should therefore be viewed as a small-deployment research instrument, not a universal labeling service.

2.2 Annotation quality and dataset traceability

Annotation-interface choices can influence produced labels and downstream models. Bauchwitz and Cummings showed that task configuration and drawing tools affected segmentation annotation quality and model performance [6]. Our current evaluation does not include human reviewers, so it cannot measure this effect. A future human study must therefore define its sampling and outcome measures in advance rather than assume that a functional interface produces high-quality judgments.

Dataset documentation also requires provenance, intended use, and limitations to be recorded [7]. Domain-specific datasets such as SUN3D demonstrate that data collection, annotation tools, and the resulting labels often need to be designed together [20]. More generally, reproducible scientific-computing practice depends on preserving the data, code, dependencies, and steps needed to reconstruct an analysis [21]. We apply that principle at the software-evaluation level by freezing source hashes, retaining raw latency observations, separating raw from summarized outputs, and stating that all measurements use synthetic metadata. These artifacts are intended to prevent a performance result from being detached from the exact code and evaluation boundary that produced it.

2.3 Access control and web hardening

The platform distinguishes administrators from annotators using a minimal form of role-based access control (RBAC). RBAC is a well-established model in which permissions are attached to roles rather than managed independently for every user [8]. Here, annotators can request and edit assigned records; administrators can inspect statistics, export data, and manage backups, but the interface keeps administrators read-only for annotation records.

Implementation choices follow current framework and defensive-development guidance. Flask documents the need to address CSRF, cookie attributes, content-security policy, resource limits, and security headers at the application or deployment layer [9]. The synchronizer-token pattern and avoidance of state-changing GET requests follow OWASP guidance [10]. Passwords are stored using Werkzeug's adaptive salted scrypt facility [11], with parameters explicitly set to N=2^17, r=8, and p=1, matching the minimum scrypt configuration listed by the OWASP Password Storage Cheat Sheet at the time of implementation [12]. Structured inputs use server-side allowlists, type checks, and length or range bounds [13]. Mapping identified weaknesses to implementation and verification artifacts also follows the traceability emphasis of the NIST Secure Software Development Framework [14]. These choices reduce identified risks but do not make the application secure against every attacker or deployment error.

3. REQUIREMENTS AND SYSTEM DESIGN

3.1 Operational requirements

Six requirements were derived from the existing workflow and the baseline audit.

Table 1. Operational requirements, implemented mechanisms, and evaluation evidence.

ID	Requirement	Implemented mechanism	Evaluation evidence
R1	Review structured image records as one unit	Keep/discard decision, text edit, normalized regions, and two controlled label lists in one record	Field round-trip test
R2	Separate annotator and administrator capabilities	Integer-valued two-role checks on protected routes	Authentication and RBAC tests
R3	Avoid cross-user duplicate allocation	One repository-level re-entrant lock around read–select–write allocation	360 in-process and 120 loopback-HTTP clean-start scenarios
R4	Preserve recoverability and exportability	Atomic JSON replacement, bounded backups, validated restore, JSON/CSV export	Corruption, export, backup, restore, and dataset-scale tests
R5	Reduce unnecessary sensitive-data exposure	Local runtime directory, no external API, externalized secrets, minimized status response, synthetic research package	Package inventory and response checks

International Journal of Computer Applications Technology and Research
Volume 15–Issue 09, 01 – 10, 2026, ISSN: -2319–8656
DOI:10.7753/IJCATR1509.1001

ID	Requirement	Implemented mechanism	Evaluation evidence
R6	Reject malformed or structurally unsafe updates	Server-side schema, bounds, type, size, path, and enumeration validation	Negative-input tests

R5 is deliberately limited. The application does not itself encrypt the disk, terminate TLS, control operating-system accounts, or define institutional retention policy. Those controls belong to deployment and governance.

3.2 Architecture

Figure 1 summarizes the architecture. A browser client communicates with Flask routes. Authentication and per-session CSRF checks precede role-aware route handling. The annotation service reads and updates records through a single repository abstraction. Data, accepted records, users, assignments, class lists, exports, and backups are stored beneath an externally configured runtime directory. No real runtime data are included in the research package.

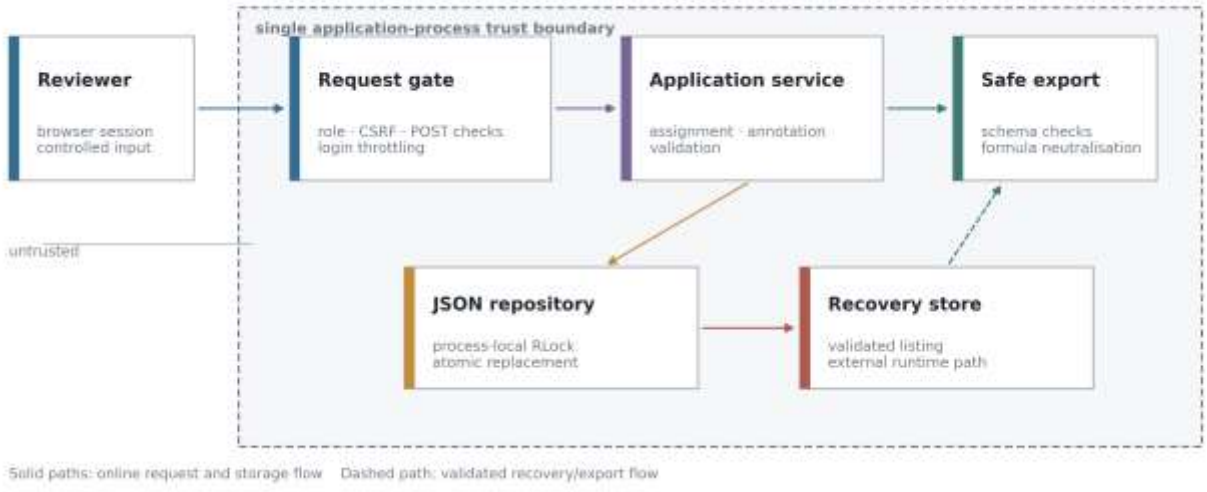

Figure 1. System architecture and the research-package boundary.

The data schema supports two historical message layouts (`conversations` and `messages`) so that editable text can be retained without a destructive migration. Each record has an identifier, one image reference, a user assignment, a tri-state review value (unmarked, keep, or discard), zero or more normalized regions, and two controlled label lists. A rectangular region is represented as `[class_id, x_center, y_center, width, height]`. The repository rejects negative class identifiers, nonnumeric coordinates, non-positive dimensions, coordinates outside [0,1], and any rectangle that extends outside the normalized image boundary. The two label lists are intentionally treated as generic enumerations in this study; no domain vocabulary is included in the artifact or evaluated in the experiments.

3.3 Assignment and persistence

An annotator requests a fixed-size batch. Within one `RLock` transaction boundary, the repository reads the master list, selects the first currently unassigned records, assigns the numeric user identifier, atomically replaces the master file, and atomically updates the user's assignment file. Usernames never determine storage paths. If the current assignment still contains unmarked records, the request is rejected to avoid silently accumulating unfinished batches.

JSON replacement uses a temporary file in the destination directory, UTF-8 serialization, flush, `fsync`, and `os.replace`. Malformed existing JSON raises an error and is not silently replaced with an empty collection. This behavior favors visible failure over hidden data loss. The transaction lock is process-local; it does not coordinate multiple WSGI workers or machines.

For review completion, records marked `True` are merged into the accepted collection by identifier and records marked `False` are removed if previously present. CSV export quotes every field and prefixes text whose first non-space character is `=`, `+`, `-`, or `@` to reduce spreadsheet-formula execution. Backups use generated names with a fixed regular expression. Restore accepts only listed backup directories, validates JSON before replacement, and restores only an allowlist of repository files.

3.4 Authentication and request protection

Registration is disabled by default. When enabled for controlled onboarding, separate administrator and annotator invitation tokens are read from configuration rather than embedded phrases. User passwords must contain at least ten characters and are stored as salted scrypt hashes. Existing plaintext entries can be migrated after one successful legacy login, after which the plaintext key is removed. This compatibility path should be disabled after migration in a production revision.

Each session receives an unpredictable CSRF token. All POST, PUT, PATCH, and DELETE requests must provide the token in a form field or custom header, and comparison uses a constant-time function. Assignment, merging, item updates, class creation, logout, backup creation, and restore are POST operations. Repeated login failures are throttled by client address within the application process. Authenticated responses set `Cache-Control: no-store`, while general responses set `X-Content-Type-Options`, `X-Frame-Options`, `Referrer-Policy`, `Permissions-Policy`, and a Content Security Policy (CSP). The present CSP retains

`unsafe-inline` for script and style compatibility and is therefore not a strict production policy.

4. EVALUATION METHOD

4.1 Frozen implementation and environment

The evaluated implementation and associated artifacts were frozen before analysis. The evaluation package contains three sequential in-process sessions, one loopback-HTTP session, and one independent-process boundary session; a manifest records source-file SHA-256 hashes and maps each summary to its raw output. Dependencies were Python 3.12.10, Flask 3.1.3, Werkzeug 3.1.8, Requests 2.33.1, pytest 9.0.3, NumPy 2.4.2, and Matplotlib 3.10.9. Experiments ran on Windows 11 with 14 logical CPUs. The three in-process sessions were started and run sequentially on this same host; they are repeated software sessions, not independent hardware environments.

All inputs were generated metadata. The endpoint and concurrency benchmarks each used 1,000 records containing fictional identifiers, relative paths to nonexistent image files, fixed placeholder messages, generated review text, and empty structured fields. The scale experiment used the same generator at 100, 1,000, 5,000, and 10,000 records. Test users were fictional and stored in temporary directories. No image file, real username, operational password, domain label vocabulary, or research dataset was read into the benchmark outputs.

4.2 Functional tests

Twelve pytest cases covered: unauthenticated access, CSRF and response headers; rejection of path-like usernames and password hashing; administrator/annotator permission boundaries; concurrent unique assignment; invalid-box rejection, valid round trip, and merge; identifier lookup; CSV formula neutralization; backup restore and traversal rejection; failure-closed handling of corrupted JSON; login throttling; page plus profile rendering; and an injected `os.replace` failure. The last test verifies that replacement failure leaves the previously committed repository bytes unchanged and removes the temporary file. The test suite was executed in each of the three in-process sessions so that its log is stored beside the measurement files.

4.3 Targeted control matrix

The baseline source audit produced 15 concrete controls: adaptive password storage, externally configured registration privilege, separation of filenames from user input, CSRF validation, POST for state changes, called validation before persistence, atomic writes, validated restore, formula neutralization, reduced status disclosure, consistent role typing, failure-closed JSON parsing, login throttling, browser security headers, and exclusion of operational accounts and images from the research package.

The controls were defined *after* inspection of the baseline and then applied to both versions. Consequently, the matrix is neither preregistered nor an unbiased comparison of overall security. It is used only to trace each identified weakness to a remediation and a check. Verification combines source patterns, route extraction, package inventory, and representative executable tests. It is not a penetration test.

4.4 Concurrency and endpoint measurements

Concurrency levels were 1, 5, 10, and 20 clients, with 30 clean-start repetitions per level and three sequentially started in-process sessions. Every scenario recreated the application, temporary repository, fictional users, and the same 1,000-record input before timing began. In each repetition, every client requested ten records and then saved one reviewed record. Integrity required the observed assignment count to equal the expected count, no identifier to appear in two users' assignments, every request to succeed, and saved fields to match the sent values. Recreating the repository prevents the repeat number, accumulated assignments, or input size from confounding the concurrency comparison.

The local Flask test client measured per-request elapsed time with a monotonic high-resolution timer. Additional endpoint measurements on the 1,000-record dataset comprised 30 single-record reads, 20 statistics requests, 10 CSV exports, five backups, and one restore per session. We retain all request-level observations but summarize the repeated runs as the median and range of each session's descriptive percentile. This treats the session, rather than thousands of correlated requests on one host, as the replication level. Because the test client bypasses the network, browser, TLS, and a production WSGI server, these values characterize only this implementation and host.

4.5 Loopback-HTTP measurements

To check whether the in-process path understated framework and transport overhead, a separate experiment ran Flask through Werkzeug's threaded WSGI server on `127.0.0.1` and sent actual HTTP requests with Requests. One conventional login and CSRF round trip verified the route path. The timed scenarios then used signed sessions for fictional accounts so that password hashing did not confound allocation and save measurements. The same four concurrency levels, 30 clean starts per level, fixed 1,000-record input, integrity criteria, and ten-record allocation were retained. Request-level P50 and P95 values describe individual HTTP calls. For scenario wall time, 10,000 bootstrap resamples of the 30 complete scenarios at each level estimated a 95% interval for the median (seed 20260825). This experiment still excludes a browser, TLS, reverse proxy, remote network, and production WSGI stack.

4.6 Dataset-size measurements

The scale experiment recreated the repository at 100, 1,000, 5,000, and 10,000 generated records. After one untimed warm-up, each size received 30 JSON loads, 30 administrator-statistics requests, 10 CSV exports, and five atomic full-file saves in each of the three sessions, producing 900 timed observations. The experiment checked record counts and response status for every operation. P50 and P95 values are descriptive; results are summarized as the median and range of the three session-level percentiles. No model is extrapolated beyond the four evaluated sizes.

4.7 Independent-process boundary test

The repository documents its `RLock` as process-local. We therefore added a negative boundary test rather than treating that statement as sufficient evidence. Each scenario created a new 1,000-record repository, then released either two or four independently spawned Python processes at the same time. Every process instantiated the unmodified `FileRepository` and called its unmodified `assign` method to request ten records. Ten clean starts were run at each process count. Because 1,000 records were available, a correct cross-process transaction could have returned 20 or 40 distinct records without exhaustion.

Integrity required all processes to return, the reported assignment rows to equal the intended total, every identifier to be unique across assignment files, and each assignment-file owner to match the final master-file owner. This was a falsification experiment, not a latency benchmark: any

International Journal of Computer Applications Technology and Research
Volume 15–Issue 09, 01 – 10, 2026, ISSN: -2319–8656
DOI:10.7753/IJCATR1509.1001

violation would disprove multi-process correctness for the tested implementation and host, whereas a finite set of passes would not prove correctness. The spawned processes, configuration, raw worker outcomes, scenario-level checks, script hash, and storage-module hash were retained.

5. RESULTS

5.1 Functional behavior and targeted controls

All 12 automated tests passed in each session (7.61, 7.81, and 8.15 s). Five field groups—review decision, editable text, normalized regions, and two controlled label lists—completed a write/read round trip. Export neutralized the test formula string, the restored repository contained the same number of records as before deliberate modification, and the injected replacement failure preserved the last committed repository.

The legacy copy satisfied 0 of the 15 audit-derived controls, whereas the hardened research copy satisfied 15 of 15 (Figure 2). This result answers RQ2 only in the narrow sense of remediation traceability. It must not be interpreted as a security score, an estimate of vulnerability prevalence, or evidence of certification.

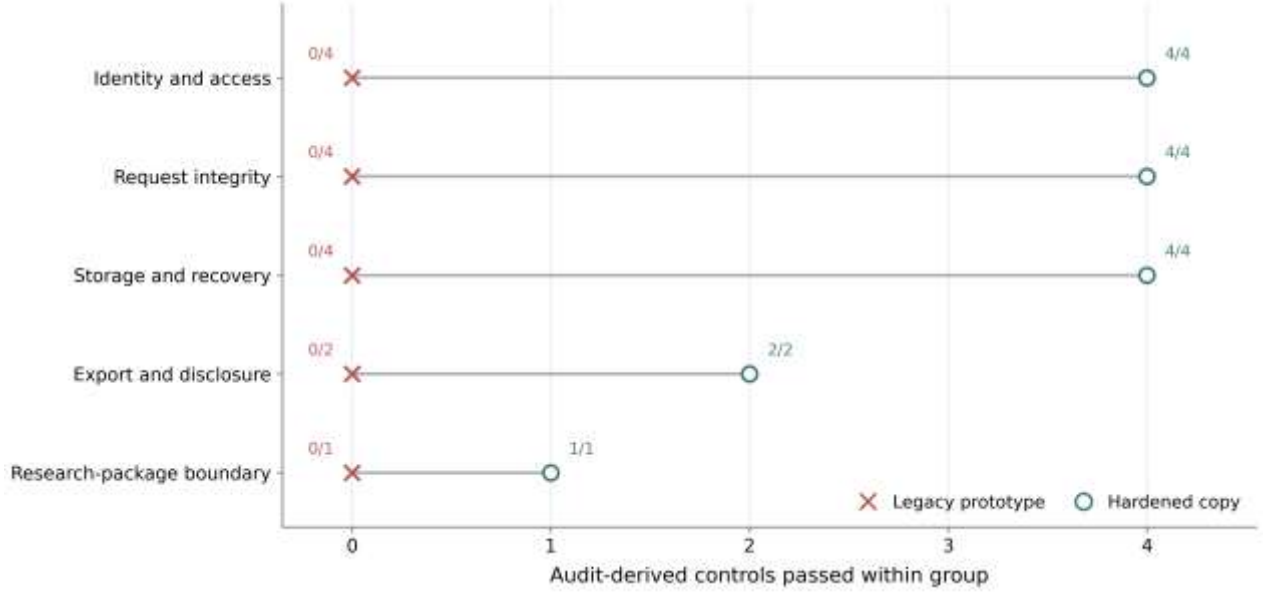

Figure 2. Audit-derived controls satisfied by concern (15 total); a remediation trace, not a security score or certification.

5.2 Assignment integrity and concurrent latency

Every one of the 360 in-process concurrency scenarios met the predefined integrity conditions. Across all repetitions and sessions, 32,400 records were allocated and no duplicate identifier was observed (Figure 3A). Figure 3B contrasts this supported scope with the independently spawned-process boundary test reported in Section 5.7. This provides repeated evidence for RQ1 under a single Python process and the tested request pattern; it is not a proof for untested schedules or failures.

Table 2. In-process assignment integrity and request P95 latency across three sessions (median and range).

Concurrent clients	Repeats across sessions	Allocated records	Duplicate IDs	Allocation success	Allocation P95 median (range), ms	Save P95 median (range), ms
1	90	900	0	100%	54.5 (38.6–54.7)	11.1 (9.4–14.7)
5	90	4,500	0	100%	218.9 (210.4–225.9)	43.7 (42.3–44.4)
10	90	9,000	0	100%	414.6 (386.2–432.7)	85.2 (83.7–85.3)
20	90	18,000	0	100%	760.2 (755.5–807.4)	157.1 (156.7–158.9)

International Journal of Computer Applications Technology and Research
Volume 15–Issue 09, 01 – 10, 2026, ISSN: -2319–8656
DOI:10.7753/IJCATR1509.1001

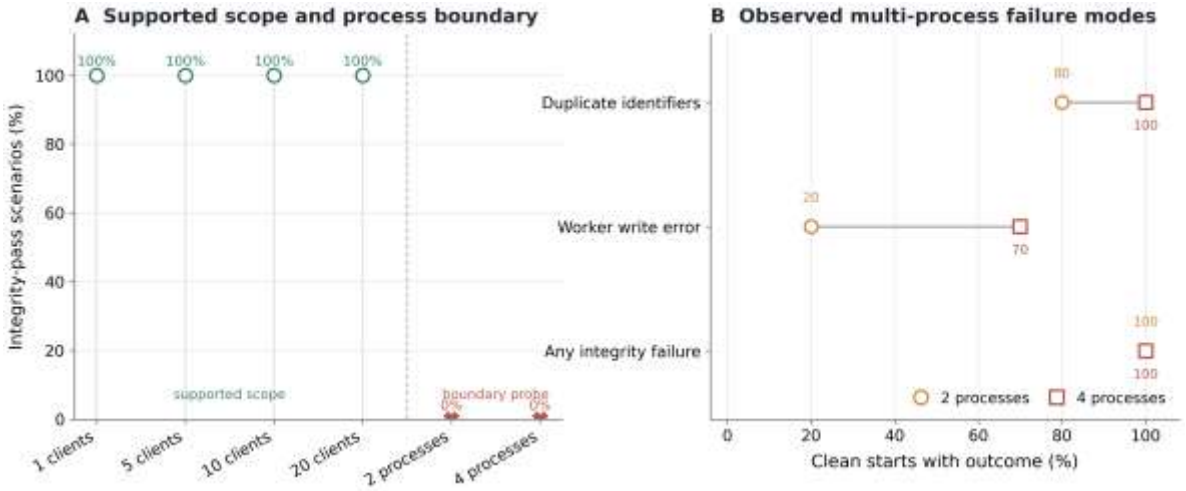

Figure 3. Scenario-level integrity in the supported single-process scope and negative multi-process boundary test. (A) Each client level had 90 clean starts; each process count had ten. (B) Failure categories overlap; "2p" and "4p" denote spawned-process counts. This falsification test is not a supported-deployment benchmark.

Both operations showed increasing latency as contention increased (Figure 4). Allocation was slower than saving because an allocation transaction scans and rewrites the shared master list as well as an assignment file. At 20 clients, allocation P95 remained below one second in this local test, but the trend shows the cost of serializing JSON read–modify–write transactions under one process lock.

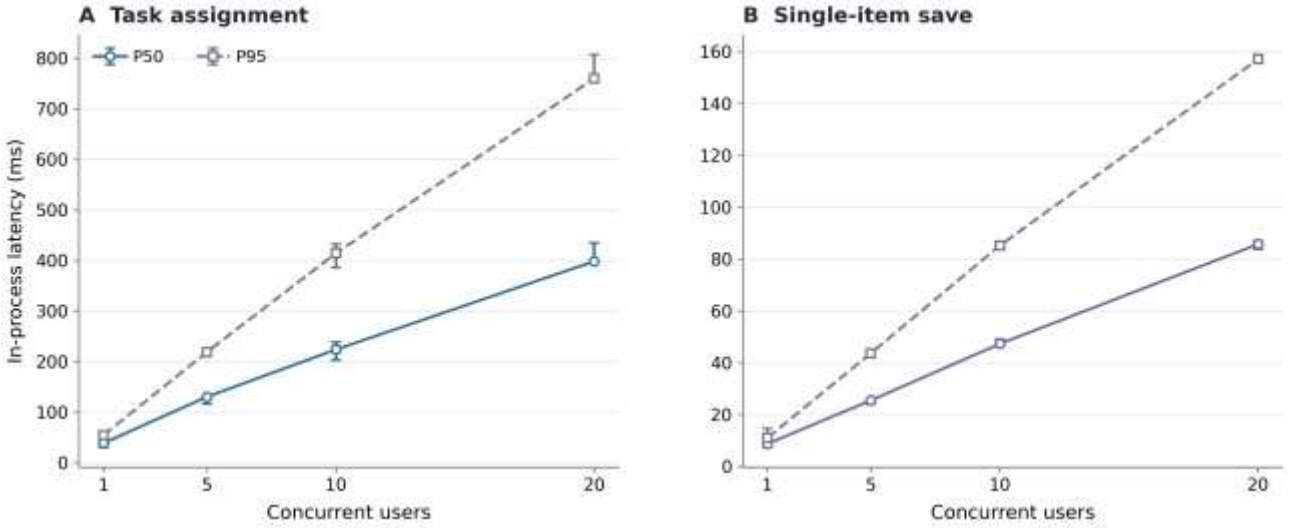

Figure 4. P50 and P95 request latency in the Flask in-process test client. Points are medians and error bars are ranges of session-level percentiles across three sequential sessions. Values exclude network, browser, TLS, and production-server overhead.

5.3 Other local endpoint measurements

On 1,000 synthetic records, the median per-session P95 was 8.0 ms for single-record retrieval (session range 7.1–9.4), 14.5 ms for administrator statistics (13.0–14.6), 13.2 ms for CSV export (12.0–14.3), and 17.0 ms for backup (16.5–18.8). Restore was run once per session; its observed values ranged from 21.4 to 40.9 ms, with a median of 33.1 ms, and each restore preserved the record count. All measured endpoint groups had 100% success.

These descriptive values answer RQ3 for the frozen host and implementation. No hypothesis test is reported: the sessions characterize run-to-run variation in one controlled environment and are neither independent samples of deployment environments nor a basis for population inference.

5.4 Loopback-HTTP sensitivity check

All 120 loopback-HTTP scenarios passed, allocating 10,800 records without a duplicate identifier. At 20 clients, request-level P95 was 913.8 ms for allocation and 202.7 ms for saving, compared with 760.2 and 157.1 ms for the medians of the three in-process session P95 values. The complete-scenario median wall times were 944.9 ms for allocation (bootstrap 95% interval 927.6–959.5 ms) and 216.9 ms for saving (210.4–223.7 ms). These values show that the in-process results do not fully represent even loopback HTTP overhead. They are a sensitivity check, not a direct estimator of remote or production latency.

5.5 Dataset-size sensitivity

All 900 scale observations passed their record-count or response-status checks. Table 3 shows that P95 latency generally increased across the tested range, with full-file atomic saving having the largest absolute cost at 10,000 records. The nonmonotonic export result between 100 and 1,000 records is retained rather than smoothed; it illustrates why the four sizes and three same-host sessions support only descriptive, bounded conclusions.

International Journal of Computer Applications Technology and Research
Volume 15–Issue 09, 01 – 10, 2026, ISSN: -2319–8656
DOI:10.7753/IJCATR1509.1001

Table 3. Median (range) of three session-level P95 values by generated dataset size. Per-session observation counts at each size were 30 for load and statistics, 10 for export, and five for save.

Records	Load P95, ms	Statistics P95, ms	CSV export P95, ms	Atomic save P95, ms
100	1.1 (1.0–1.3)	5.1 (2.7–5.4)	12.1 (8.4–12.5)	9.6 (6.6–15.6)
1,000	3.4 (3.2–4.6)	6.8 (6.5–10.9)	10.0 (9.4–11.0)	30.8 (25.4–32.6)
5,000	21.4 (20.1–22.5)	35.5 (34.5–37.5)	43.5 (41.1–43.8)	110.1 (107.3–201.8)
10,000	46.5 (45.8–56.4)	68.5 (64.5–69.9)	75.4 (64.3–82.6)	185.6 (176.5–187.6)

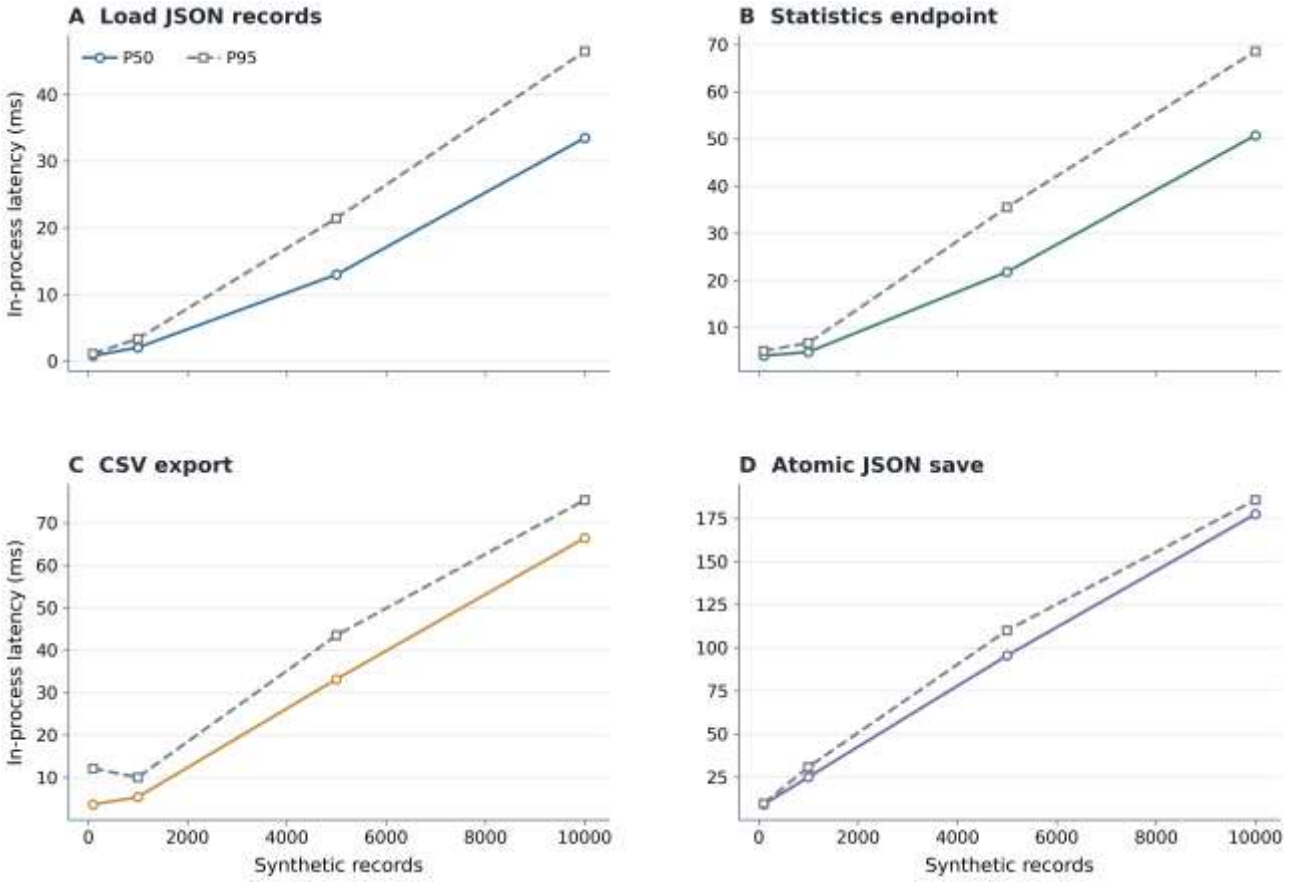

Figure 5. Dataset-size sensitivity of JSON-backed operations. Values exclude image loading, network transfer, browser rendering, and production-server overhead.

The result answers RQ4 by making the deployment trade-off measurable. At 10,000 records, the median session-level P95 remained below 100 ms for loading, statistics, and export, while atomic save reached 185.6 ms. These same-host observations are compatible with occasional full-file persistence in the evaluated workflow, but the growth and rewrite-on-update design argue against treating JSON as a path to larger or multi-worker deployment.

5.6 Independent-process falsification result

None of the 20 independent-process scenarios met the integrity criterion (Figure 3B and Table 4). Eighteen scenarios produced duplicate identifiers across assignment files and corresponding assignment-to-master ownership mismatches. Nine scenarios contained at least one worker write error, comprising 12 `PermissionError` outcomes from competing replacements on this Windows host. Across all scenarios, the workers reported 480 of 600 intended assignment rows; 280 of the reported rows duplicated an identifier assigned by another process, and 280 assignment rows disagreed with the owner retained in the final master file. Failure categories overlap: a scenario could contain both a write error and duplicated assignments.

Table 4. Negative boundary test with independent spawned processes sharing one generated JSON repository. Results characterize this implementation and Windows host; they are not an estimate of failure prevalence on other operating systems or schedules.

Processes	Clean starts	Integrity passes	Starts with duplicate IDs	Starts with worker error	Intended / reported assignment rows	Duplicate rows	Owner mismatches
2	10	0	8	2	200 / 180	80	80
4	10	0	10	7	400 / 300	200	200

The result completes the boundary component of RQ1. The single-process `RLock` prevented overlapping in-process transactions in the positive experiments, but it did not become a cross-process transaction merely because each final JSON replacement was atomic. Atomic replacement protected an individual file from partial serialization; it did not make the multi-file read–modify–write sequence isolated.

6. DISCUSSION

6.1 What the evaluation establishes

The combined tests establish that the hardened copy performs the intended narrow workflow with synthetic metadata, enforces the tested role boundaries, rejects representative malformed inputs, and avoids duplicate allocation in the exercised single-process concurrency cases. The fixed-input concurrency design separates client contention from dataset size, while the independent scale experiment exposes the cost of full-file JSON operations. The negative boundary test then converts a documented restriction into an observed failure mode: atomic file replacement and a process-local lock did not isolate cross-process assignment transactions. The artifact trail links paper values to raw observations, dependency versions, and source hashes. These are useful properties for a laboratory tool whose primary risk is silent corruption or uncontrolled handling of a modest local dataset.

The study also illustrates why functional completion and engineering hardening should be evaluated together. A working annotation interface can still expose passwords, mix state changes with GET requests, accept malformed boxes, or overwrite a corrupted file. Conversely, adding security headers does not establish correct task allocation. The evaluation therefore keeps functional, integrity, performance, and targeted-control evidence separate rather than collapsing them into a single score.

6.2 Relationship to general-purpose platforms

The platform is not intended to outperform CVAT, Label Studio, VIA, or OneLabeler. Those systems are preferable when a project needs mature deployment options, broad annotation modalities, project administration, plug-ins, model-assisted labeling, or large-scale collaboration. The value of the present implementation is compatibility with one existing structured-record schema, together with a small auditable code path and an evaluation package that excludes operational images and domain labels.

This trade-off has a practical implication. A group choosing between migration and hardening should first test whether its workflow can be expressed directly in a maintained general-purpose platform. A custom system is justified only when the migration and customization burden is material and the group accepts responsibility for maintenance, threat modeling, backups, and testing.

6.3 Deployment guidance

The evaluated JSON backend should be deployed with one application process only. The independent-process experiment directly observed duplicate assignment, ownership mismatch, and competing-replacement errors when multiple workers each held their own `RLock`. A multi-worker or multi-host installation must replace JSON transactions with a database that provides transactional row or task claiming, or introduce a tested inter-process lock spanning the complete multi-file transaction. Production deployment also requires HTTPS at a reverse proxy, a stable secret key, restricted operating-system permissions, encrypted and access-controlled backups, centralized rate limiting, logging and monitoring, and a tested retention/deletion policy. Inline JavaScript and CSS should be moved to static files so that `unsafe-inline` can be removed from CSP.

6.4 Limitations and future work

First, all experiments used one Windows workstation. The HTTP sensitivity run covered only a threaded Werkzeug server on loopback and still excluded a browser, TLS, reverse proxy, remote network, image decoding, and a production WSGI stack. The three in-process sessions measure run-to-run variation, not between-machine generalizability. Second, the concurrency input was fixed at 1,000 records and the separate scale experiment stopped at 10,000; both used synthetic metadata without image files. Third, positive concurrency evidence remains bounded to one process and the exercised scenarios. The independent-process experiment establishes a failure on this Windows host but does not estimate failure frequency under other process schedules, operating systems, network filesystems, or worker models. Fourth, the 15 controls were derived after auditing the baseline and cover only identified concerns. They do not measure undiscovered vulnerabilities, operating-system security, dependency compromise, or malicious authenticated users. Fifth, no penetration test, power-loss or `fsync` fault injection, process crash during a multi-file transaction, or cross-office-suite formula-injection study was conducted. The injected `os.replace` failure covers only one atomic-write failure point.

Most importantly, no real reviewer participated. The study therefore provides no evidence about learning time, completion time, cognitive load, accessibility, inter-reviewer agreement, or the quality of text and region judgments. A future evaluation should obtain the necessary institutional and participant approvals, use a predeclared sampling and agreement protocol, compare the task-specific interface with a configured general-purpose platform, and separate interface effects from reviewer expertise. Only after that study would claims about review efficiency or quality be justified.

7. CONCLUSION

We presented a lightweight local platform for structured image-record review and evaluated an isolated, hardened research copy without using operational accounts, image files, domain labels, or research datasets. The copy passed 12 functional tests in each of three sessions, addressed all 15 audit-derived controls, and allocated 32,400 synthetic records without duplicate identifiers across 360 in-process concurrency scenarios. A separate 120-scenario loopback-HTTP check retained allocation integrity, and three scale sessions quantified 900 JSON-backed observations from 100 to 10,000 records. In contrast, none of 20 independent-process boundary scenarios met the integrity criterion; duplicate assignments, master-file ownership mismatches, and competing-replacement errors empirically rejected multi-process use of the current repository. The observed performance supports only the bounded single-host, single-process configuration and reveals the contention, growth, and deployment costs of a locked, rewrite-based JSON repository. The principal result is an engineering artifact and evidence trail—not a new algorithm, a domain dataset, a general annotation benchmark, or a security certification. Moving beyond this boundary requires a transactional backend, production deployment tests, and a properly approved human-centered review study.

DATA AND CODE AVAILABILITY

The sanitized source code, tests, synthetic-data generator, raw measurement files, summaries, and plotting script are available from the corresponding author upon reasonable request. Operational account files, image files, domain vocabularies, and research datasets are not included.

ETHICS AND PRIVACY STATEMENT

The evaluation reported in this paper used generated metadata, fictional users, and no image files. It involved no human participants and reports no participant outcomes. No operational or research dataset was copied into the research package or used for the measurements. Any future study involving human reviewers or real images must undergo the institution's applicable ethics, consent, privacy, and data-governance review before data collection or publication.

FUNDING

This study was not supported by any grants from funding bodies in the public, private, or not-for-profit sectors.

CONFLICT OF INTEREST

The author declares no conflict of interest.

AUTHOR CONTRIBUTIONS

Zhendong Wang: Conceptualization, Software, Methodology, Validation, Formal analysis, Visualization, Writing—original draft, Writing—review and editing.

DECLARATION OF GENERATIVE AI USE

During the preparation of this work, the author used OpenAI Codex to assist source-code review, test drafting, literature discovery, and language editing. The author reviewed and revised all outputs, verified the reported results and references, and accepts full responsibility for the content of the article.

REFERENCES

[1] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman, "LabelMe: A database and web-based tool for image annotation," *International Journal of Computer Vision*, vol. 77, pp. 157–173, 2008, doi: 10.1007/s11263-007-0090-8.

[2] A. Dutta and A. Zisserman, "The VIA annotation software for images, audio and video," in *Proceedings of the 27th ACM International Conference on Multimedia*, 2019, pp. 2276–2279, doi: 10.1145/3343031.3350535.

[3] CVAT.ai Corporation, "CVAT Documentation," 2026. [Online]. Available: https://docs.cvat.ai/docs/ (accessed Aug. 24, 2026).

[4] M. Tkachenko, M. Malyuk, A. Holmanyuk, and N. Liubimov, "Label Studio: Data labeling software," 2020–2025. [Online]. Available: https://github.com/HumanSignal/label-studio (accessed Aug. 24, 2026).

[5] Y. Zhang, Y. Wang, H. Zhang, B. Zhu, S. Chen, and D. Zhang, "OneLabeler: A flexible system for building data labeling tools," in *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022, Article 93, pp. 1–22, doi: 10.1145/3491102.3517612.

[6] B. R. Bauchwitz and M. Cummings, "Task configuration impacts annotation quality and model training performance in crowdsourced image segmentation," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2025, pp. 6646–6656, doi: 10.1109/WACV61041.2025.00647. [Online]. Available: https://openaccess.thecvf.com/content/WACV2025/html/Bauchwitz_Task_Configuration_Impacts_Annotation_Quality_and_Model_Training_Performance_in_WACV_2025_paper.html

[7] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. Daumé III, and K. Crawford, "Datasheets for datasets," *Communications of the ACM*, vol. 64, no. 12, pp. 86–92, 2021, doi: 10.1145/3458723.

[8] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996, doi: 10.1109/2.485845.

[9] Pallets Projects, "Security Considerations—Flask Documentation 3.1.x," 2026. [Online]. Available: https://flask.palletsprojects.com/en/stable/web-security/ (accessed Aug. 24, 2026).

[10] OWASP Foundation, "Cross-Site Request Forgery Prevention Cheat Sheet," 2026. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html (accessed Aug. 24, 2026).

[11] Pallets Projects, "Security Helpers—Werkzeug Documentation 3.1.x," 2026. [Online]. Available: https://werkzeug.palletsprojects.com/en/stable/utils/#module-werkzeug.security (accessed Aug. 24, 2026).

[12] OWASP Foundation, "Password Storage Cheat Sheet," 2026. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (accessed Aug. 24, 2026).

[13] OWASP Foundation, "Input Validation Cheat Sheet," 2026. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html (accessed Aug. 24, 2026).

[14] M. Souppaya, K. Scarfone, and D. Dodson, *Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities*, NIST Special Publication 800-218, 2022, doi: 10.6028/NIST.SP.800-218.

[15] S. Kunwar, "Annotate-Lab: Simplifying image annotation," *Journal of Open Source Software*, vol. 9, no. 103, Article 7210, 2024, doi: 10.21105/joss.07210.

[16] M. Cormier, F. Röpke, T. Golda, and J. Beyerer, "Interactive labeling for human pose estimation in surveillance videos," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2021, pp. 1649–1658, doi: 10.1109/ICCVW54120.2021.00190.

[17] C. Xu, B. Dong, N. Stier, C. McCully, D. A. Howell, P. Sen, and T. Höllerer, "Interactive segmentation and visualization for tiny objects in multi-megapixel images," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 21447–21452.

[18] P. Stenetorp, S. Pyysalo, G. Topić, T. Ohta, S. Ananiadou, and J. Tsujii, "brat: A web-based tool for NLP-assisted text annotation," in *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, 2012, pp. 102–107.

[19] J.-C. Klie, M. Bugert, B. Boullosa, R. Eckart de Castilho, and I. Gurevych, "The INCEpTION platform: Machine-assisted and knowledge-oriented interactive annotation," in *Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations*, 2018, pp. 5–9.

[20] J. Xiao, A. Owens, and A. Torralba, "SUN3D: A database of big spaces reconstructed using structure from motion and object labels," in *Proceedings of the IEEE International*

Conference on Computer Vision, 2013, pp. 1625–1632, doi: 10.1109/ICCV.2013.458.

[21] G. Wilson, J. Bryan, K. Cranston, J. Kitzes, L. Nederbragt, and T. K. Teal, "Good enough practices in scientific computing," *PLOS Computational Biology*, vol. 13, no. 6, Article e1005510, 2017, doi: 10.1371/journal.pcbi.1005510.