IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154129

# LUMEX: A Web-Based Image Enhancement Studio Using Python, Flask, and Pillow

**Prataparao Charith[1], K.Varun[2], Yash Kumar[3], Dr. H. Mary Shyni[4]**

Student, Department of Computer Science and Engineering,

SRM Institute of Science and Technology Vadapalani[1]

Student, Department of Computer Science and Engineering,

SRM Institute of Science and Technology Vadapalani[2]

Student, Department of Computer Science and Engineering,

SRM Institute of Science and Technology Vadapalani[3]

Assistant Professor, Department of Computer Science and Engineering,

SRM Institute of Science and Technology Vadapalani Campus Chennai[4]

**Abstract**—Digital image processing is a critical component of modern software architecture, yet existing solutions frequently oscillate between prohibitive complexity for casual users and expensive proprietary licensing. This paper introduces LUMEX, a lightweight, browser-based studio for image enhancement developed using Python-Flask (≥2.3.0) with Pillow (≥10.0.0) and NumPy (≥1.24.0). The proposed system integrates a modular pipeline encompassing four tone controls (brightness, contrast, saturation, sharpness), eleven specialized filters, and high-fidelity geometric transformations utilizing LANCZOS resampling. The frontend, implemented as a single-page application with a dark-themed interface, supports drag-and-drop upload, real-time parameter adjustment, and a three-mode image viewer (Original / Enhanced / Split). Architecturally, LUMEX exposes four stateless REST endpoints over HTTP. Experimental results demonstrate sub-300 ms processing latency for HD images on commodity hardware, with JPEG output quality configurable from 10 to 100 (default: 92). By utilizing a fully self-hosted, open-source stack, LUMEX delivers a secure and professional-grade alternative to cloud-based editors that compromise user data privacy.

**Keywords**—Digital image processing, Flask, Image enhancement, LANCZOS resampling, Pillow, Python, REST API, Web application.

## I. INTRODUCTION

The ubiquity of high-resolution smartphone photography and social media has escalated the demand for accessible yet powerful image manipulation tools. A significant gap persists between high-end professional software—which demands extensive computational resources and proprietary licensing—and free online editors that impose privacy risks by transmitting user imagery to remote servers.

LUMEX (derived from lumen and excellence) was developed to address this disparity. Deployed locally at http://localhost:5000, it provides a self-contained web environment where users can perform complex enhancements without any external cloud dependency. The system offers a dark-themed single-page application (SPA) built on three open-source packages—Flask, Pillow, and NumPy—each pinned to minimum production-grade versions as documented in requirements.txt.

The remainder of this paper is organized as follows. Section II reviews relevant literature. Section III details system architecture, the enhancement pipeline, and the frontend interface. Section IV presents experimental results, and Section V concludes with future directions.

## II. LITERATURE REVIEW

The theoretical foundation for the enhancement algorithms in LUMEX is rooted in spatial filtering and intensity transformation techniques pioneered by Gonzalez and Woods [1]. Histogram equalization follows established models for redistributing pixel intensities to optimize dynamic range.

  

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154129

The processing engine utilizes the Pillow library (≥10.0.0) [2], whose ImageEnhance module applies linear interpolation between a degenerate image and the original to deliver precise tonal control. LANCZOS resampling, identified by Turkowski [6] as superior to bilinear interpolation for suppressing aliasing, is used exclusively for geometric resize operations. The stateless REST design adheres to Fielding's architectural constraints [4], ensuring the server retains no client session state between requests and simplifying horizontal scalability.

## III. METHODOLOGY

### *A. Software Stack and Project Layout*

LUMEX depends on three packages declared in requirements.txt, summarized in Table IV. Flask handles HTTP routing and multipart request parsing. Pillow provides all image I/O, enhancement primitives, and kernel-based filter operations. NumPy supplies float64 array operations used exclusively in the sepia channel-mapping transform. The project follows a minimal directory layout: app.py at the root contains both the Flask application factory and the full image-processing logic; templates/index.html is the sole frontend template; and static/uploads/ and static/outputs/ are auto-created at startup via os.makedirs.

**TABLE IV. Python Package Dependencies (from requirements.txt)**

| Package | Min. Version | Role |
|---|---|---|
| Flask | ≥2.3.0 | HTTP routing, request/response handling |
| Pillow | ≥10.0.0 | Image I/O, enhancement, filtering, transforms |
| NumPy | ≥1.24.0 | Float64 matrix multiply for sepia channel mapping |

### *B. System Architecture and REST API*

LUMEX follows a stateless REST architecture. All image computation occurs on the backend; the frontend is a thin client that communicates exclusively via fetch() calls and JSON payloads. This separation ensures consistent results regardless of the client's hardware. The application exposes four HTTP endpoints, listed in Table I. The /upload endpoint validates file extensions against an allowlist of six types (png, jpg, jpeg, webp, bmp, tiff), persists the image under a UUID-derived filename to prevent collisions, and returns a JSON object containing filename, dimensions, color mode, and size in kilobytes. The /enhance endpoint accepts the stored filename and all parameter values in a JSON body, executes the pipeline, and returns the output URL together with updated metadata. The /download endpoint routes requests to either the uploads or outputs directory based on the enhanced_ filename prefix.

**TABLE I. REST API Endpoints**

| Method | Endpoint | Description |
|---|---|---|
| GET | / | Serve the frontend SPA |
| POST | /upload | Validate & store image; return metadata JSON |
| POST | /enhance | Apply pipeline; return processed image URL |
| GET | /download/<f> | Serve original or enhanced file as attachment |

### *C. Tone Adjustment Pipeline*

Tone adjustments are applied in fixed sequential order—brightness, contrast, saturation, sharpness—so each stage operates on the fully corrected output of its predecessor. All parameters are floating-point factors anchored at 1.0 (neutral). Any factor equal to 1.0 is skipped to avoid redundant computation. The frontend exposes these as HTML range sliders with the bounds and granularity specified in Table III; the Sharpness slider extends to 5.0 to accommodate strong unsharp-mask effects, while Saturation begins at 0.0 to allow full desaturation.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154129

**TABLE III. Frontend Slider Parameter Ranges (from index.html)**

| Control | Min | Max | Step | Default |
|---|---|---|---|---|
| Brightness | 0.1 | 3.0 | 0.05 | 1.0× |
| Contrast | 0.1 | 3.0 | 0.05 | 1.0× |
| Saturation | 0.0 | 3.0 | 0.05 | 1.0× |
| Sharpness | 0.0 | 5.0 | 0.1 | 1.0× |
| Blur Radius | 0.5 | 10.0 | 0.5 | 2.0 px |
| JPEG Quality | 10 | 100 | 1 | 92 |

*D. Filter Application*

After tone correction, exactly one of eleven named filter modes is applied per request:

1) **Convolution filters**: Blur (GaussianBlur, radius 0.5–10 px, default 2 px, exposed as a secondary slider only when the Blur mode is active), Sharpen (SHARPEN kernel), Edge (FIND_EDGES), Emboss (EMBOSS), Smooth (SMOOTH_MORE), and Detail (DETAIL).

2) **Pixel transforms**: Grayscale (single-channel conversion followed by RGB promotion), Invert (bitwise complement via ImageOps.invert), Auto-Contrast (linear histogram stretch via ImageOps.autocontrast), and Equalize (full histogram equalization via ImageOps.equalize).

3) **Sepia**: Grayscale conversion is followed by a float64 NumPy matrix multiply. The 3×3 coefficient matrix (Table II) maps each input channel linearly to the three output channels, with each result clipped to [0, 255] and cast to uint8 before reconstruction as a Pillow Image.

**TABLE II. Sepia Channel Transform Coefficients**

| Input Ch. | $R_2$ | $G_2$ | $B_2$ |
|---|---|---|---|
| $R_1$ | 0.393 | 0.769 | 0.189 |
| $G_1$ | 0.349 | 0.686 | 0.168 |
| $B_1$ | 0.272 | 0.534 | 0.131 |

*E. Geometric Transformations*

Three transform classes follow the filter stage. Rotation accepts a cumulative integer degree value (incremented in ±90° steps via the frontend buttons) and applies img.rotate(−θ, expand=True), preserving all content when the canvas changes aspect ratio. Horizontal and vertical flips use ImageOps.mirror and ImageOps.flip respectively, toggled independently as boolean state in the frontend. Resize accepts explicit integer width and height values (1–8000 px per axis) and applies img.resize((w, h), Image.LANCZOS).

*F. Output Encoding and Format Handling*

Before the pipeline begins, the loaded image is normalized to RGB via convert('RGB'), resolving RGBA transparency and mode incompatibilities for all six supported input types. The processed image is encoded as JPEG with a quality factor configurable from 10 to 100 (range slider, default 92, step 1). Each output file receives the prefix enhanced_ followed by a fresh UUID4 hex string, which the /download endpoint uses to determine the source directory.

*G. Frontend Interface*

The SPA is delivered as a single Jinja2 template (index.html, 1085 lines) with all CSS and JavaScript inlined. The visual design employs a dark color palette defined through CSS custom properties—background #0a0a0f, surface #111118, panel #16161f—with a subtle 40×40 px grid overlay rendered via a CSS pseudo-element. Three typefaces are loaded from Google Fonts: Bebas Neue for the logotype, DM Sans (300/400/500 weights) for body UI, and JetBrains Mono for monospace elements including the stats bar and section labels.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.154129

The workspace uses a CSS Grid layout divided into a fixed 320 px sidebar and a fluid main canvas panel. The sidebar is organized into four labeled sections—Tone, Filters, Transform, and Output—each separated by a 1 px border. The canvas area supports drag-and-drop upload (ondragover / ondrop handlers) as well as click-to-browse via a hidden file input. Once an image is loaded, three view modes are available—Original, Enhanced, and Split—toggled by a button group. The Split view renders both images side by side in a two-pane flex layout with overlay labels. A persistent stats bar at the bottom of the canvas reports STATUS, ORIGINAL metadata, and OUTPUT metadata using JetBrains Mono at 0.65 rem. Transient user feedback (upload success, processing errors) is delivered via a slide-in toast component positioned in the lower-right corner.

## IV. RESULTS AND ANALYSIS

Testing was conducted on twenty representative images spanning portraits, low-light landscapes, and synthetic graphics, covering all six permitted extensions (png, jpg, jpeg, webp, bmp, tiff). RGBA-to-RGB normalization correctly preserved visual content for all transparent PNG inputs. The Split-view comparison confirmed accurate spatial alignment between original and enhanced canvases across all transform combinations.

Average processing latency was measured per pipeline stage on commodity hardware. For HD (1920×1080) inputs, the complete tone-adjustment and filter pass completed in under 261 ms, satisfying near real-time requirements for web interaction. LANCZOS geometric resize added negligible overhead for proportional downscaling but approached the 300 ms threshold for large upscaling factors. JPEG output at quality 92 maintained a mean PSNR above 42 dB relative to lossless PNG baselines, confirming high fidelity suitable for professional workflows. No banding or clipping artifacts were observed in sepia-filtered outputs across the full test set.

## V. CONCLUSION

LUMEX demonstrates that professional-grade image enhancement can be delivered through a minimal open-source stack. A stateless RESTful backend (Flask + Pillow + NumPy) paired with a self-contained dark-themed SPA provides sub-300 ms HD latency, eleven filter modes, configurable JPEG quality, and a split-view comparison workflow—without any cloud dependency or proprietary licensing. The four-endpoint REST contract and UUID-based file management ensure the system is stateless, collision-free, and straightforward to deploy.

Future work will explore non-destructive editing history via a client-side undo stack, batch processing through an asynchronous task queue, and deep-learning-based denoising to extend image quality beyond the ceiling of classical spatial filters.

## ACKNOWLEDGMENT

The authors acknowledge the Department of Computer Science and Engineering at SRM Institute of Science and Technology for providing the research infrastructure. Disclosure: The authors utilized OpenAI ChatGPT-4o to assist with the grammatical polishing and structural alignment of the manuscript text.

## REFERENCES

[1] R. C. Gonzalez and R. E. Woods, Digital Image Processing, 4th ed. Pearson, 2018.

[2] A. Clark, "Pillow (PIL Fork) Documentation," Release 10.0.0, 2023. [Online]. Available: https://pillow.readthedocs.io

[3] M. Grinberg, Flask Web Development: Developing Web Applications with Python, 2nd ed. O'Reilly Media, 2018.

[4] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, Univ. of California, Irvine, 2000.

[5] A. Holovaty and J. Kaplan-Moss, The Definitive Guide to Django, 2nd ed. Apress, 2009.

[6] K. Turkowski, "Filters for Common Resampling Tasks," in Graphics Gems, A. Glassner, Ed. Academic Press, 1990, pp. 147–165.

[7] C. Poynton, Digital Video and HD: Algorithms and Interfaces, 2nd ed. Morgan Kaufmann, 2012.

[8] NumPy Development Team, "NumPy User Guide," Release 1.26, 2024. [Online]. Available: https://numpy.org/doc/stable/