International Journal of Computational Research in Science and Technology
E-ISSN: 3108-0871, Volume 2 Issue 5
Sep-Oct-2026, PP 01-23

Face Gesture Stylizer: Real-Time Facial and Hair Restyling Using Hand Gesture Recognition and Computer Vision

Puja Gupta[1] Srijan Mohan[2] Rishab Patidar[3]
Department of Information Technology
Shri G.S. Institute of Technology and Science, Indore
Indore, India

Abstract—Face Gesture Stylizer is a webcam-based computer-vision application that applies artistic transformations to a user's face and hair while retaining the natural appearance of the hands, neck, clothing and background. The application uses hand gestures to control the live effect: a right or left swipe changes the selected style, an upward or downward swipe adjusts blend intensity, and a sustained thumb–index pinch requests a photograph capture. The browser workflow uses MediaPipe Tasks Vision for hand landmark detection and selfie multiclass segmentation. A gesture engine interprets landmark movement over time, while a compositor applies the selected effect through an estimated head mask and preserves the hand region as a foreground object. The interface provides 24 built-in artistic styles and an optional FLUX.2 [klein] generative-editing path when a compatible backend is configured. The supplied screenshots demonstrate the live interface and three selected styles. Instantaneous frame-rate readings of 15–19 FPS are visible in the captures; these are observations rather than controlled benchmark results**.**

Index Terms—computer vision, hand gesture recognition, MediaPipe, semantic segmentation, facial stylization, image compositing, real-time interaction, FastAPI**.**

I. Introduction

1. Background

Camera-based visual effects have become common in communication, entertainment and social-media applications. A typical filter interface asks the user to select a style, adjust a slider or press a capture button. These controls are familiar, but they require repeated interaction with a mouse, keyboard or touch screen. A camera-based gesture interface offers an alternative: the user can control the application through deliberate hand movements while remaining in front of the live preview.

Hand gestures provide a natural way to express simple commands. A horizontal movement can indicate navigation, a vertical movement can indicate adjustment, and a pinch can indicate selection or confirmation. A computer-vision system must nevertheless distinguish an intentional gesture from ordinary movement. It needs to detect the hand, locate important points, track those points across multiple frames and apply timing rules that prevent repeated commands.

A second challenge is the region to which an artistic effect is applied. Full-frame filters change the face, clothing and background together. In portrait stylization, this may be undesirable because the intended transformation is usually limited to the face and hair. Semantic segmentation can estimate the pixels belonging to these regions, and image compositing can combine the transformed region with the original frame.

Camera-based creative applications combine a continuous visual input with controls that must respond while the user is moving. Unlike a conventional image editor, a live stylizer cannot assume that the subject will remain at one fixed position or that the user will pause to make a selection. The software must repeatedly acquire frames, estimate the relevant regions, apply the selected transformation and display the result with sufficiently low delay to preserve the feeling of direct interaction.

The Face Gesture Stylizer project brings together two different forms of visual understanding. Hand landmark detection describes the position of key points on the hand and provides the information needed to interpret movements such as swipes and pinches. Semantic segmentation estimates the class of each pixel in the camera image and helps distinguish the head

region from the body, clothing and background. These outputs serve different purposes: hand landmarks support interaction, while segmentation supports selective image editing.

The separation of interaction and rendering is important to the design. A right-hand swipe should generate a navigation event that updates the selected style; it should not directly alter image pixels. Similarly, an upward or downward movement should update the blend-intensity state. The style engine then produces the selected artistic treatment, and the compositor decides where that treatment is visible. This modular arrangement makes the application easier to understand, test and extend.

A further design consideration is visual layering. The hand may cross in front of the face while the user is issuing a command. If the stylized face layer is drawn over the hand, the result contradicts the natural foreground-background order. The project therefore uses hand information to preserve an approximate hand region above the stylized layer. The quality of this behaviour depends on landmark visibility and the estimated hand region, so it is an engineering feature with practical limitations rather than a perfect occlusion solution.

The application combines automatic and manual interaction. Gesture control is useful when the user is positioned in front of the camera and the hand is visible. The style carousel, buttons and intensity slider provide alternatives when a gesture is missed or when the user prefers conventional controls. This dual-control approach improves recoverability and makes the demonstration less dependent on ideal camera conditions.

The live interface presents 24 built-in artistic styles and shows the selected style, blend intensity, frame-rate indicator and gesture instructions. The supplied screenshots demonstrate Pop Art Comic, Anime & Manga and Urban Graffiti. They also show a left-swipe feedback notification and instantaneous frame-rate readings. These materials are useful as qualitative evidence of the interface, but they do not replace a controlled evaluation of recognition accuracy, mask quality or sustained performance.

II. RELATED WORK AND TECHNOLOGY

1. Hand Landmark Detection

MediaPipe provides reusable perception components for real-time applications [1]. Its Hand Landmarker represents a detected hand through 21 key points, including the wrist, finger joints and fingertips [2]. This representation is suitable for gesture interfaces because it describes the hand pose without requiring the application to reason directly over every image pixel. Comparing landmark coordinates over successive frames makes it possible to estimate hand movement and fingertip distance.

A single landmark frame describes pose, not motion. Swipe recognition therefore requires a temporal history. A reference point such as the palm centre can be tracked over a short interval, and its displacement can be compared with horizontal and vertical thresholds. Cooldown timing and directional consistency checks are commonly useful to avoid repeated events caused by hand jitter or the natural return movement after a swipe.

MediaPipe is relevant to this project because it packages machine-learning perception tasks in a form that can be integrated into real-time applications. The Hand Landmarker task returns a structured set of points rather than a single hand bounding rectangle. This representation is useful for gesture interaction because the relative locations of the wrist, joints and fingertips provide clues about both hand pose and movement. The project uses these landmarks as input to its gesture logic; the actual command is determined by comparing landmark positions over time, not by treating one pose as a complete swipe.

Landmark-based interaction has practical constraints. A hand may be partially outside the camera frame, hidden by another object or blurred by fast movement. Landmark coordinates can also fluctuate slightly between adjacent frames. A robust gesture controller therefore needs a short temporal history, movement thresholds and a cooldown or state-reset rule. These are application-level decisions layered on top of the landmark detector and are necessary to convert continuous tracking output into discrete commands.

2. Semantic Segmentation

Image segmentation assigns a category or confidence value to individual pixels. Unlike a bounding box, a segmentation mask can follow the approximate shape of the target region. MediaPipe's selfie multiclass segmentation task distinguishes categories such as hair, face-skin, body-skin, clothes and background [3]. Combining hair and face-skin confidence maps gives the application a practical estimate of the head region, while excluding body-skin reduces the risk of stylizing the neck or hands.

Segmentation differs from landmark detection because it produces a spatial description of the image. A face detector or landmark model may describe a face through a box or a collection of points, whereas a segmentation model estimates which pixels belong to categories such as hair, face skin, body skin, clothes and background. This pixel-level output is valuable when an effect should follow the approximate outline of the head rather than cover a rectangular region. In this project, the hair and face-skin regions are combined to form an estimated head mask.

Segmentation confidence is not equivalent to a perfect boundary. Hair strands, hats, glasses, shadows, similar background colours and motion blur can make the estimated edge uncertain. The project's use of a mask should therefore be understood as a practical region-selection technique. The mask may need resizing, smoothing or fallback handling before it is suitable for compositing, and visual inspection remains important around the hairline, ears and jaw.

3. Image Processing and Compositing

OpenCV provides image-processing operations such as resizing, colour conversion, contour processing and image encoding [4]. NumPy arrays represent image channels and masks. In a masked compositor, each output pixel is formed from the original frame and the transformed image according to a mask value and a blend-strength parameter. A soft mask creates a gradual transition at the boundary and reduces the appearance of a hard cut line.

Image compositing provides the mathematical link between segmentation and the final portrait effect. Given an original frame, a stylized frame, a mask and a blend-intensity value, the output can be formed as a weighted combination of the original and transformed pixels. A soft mask gives partial weights near uncertain boundaries and can reduce a hard cut-out appearance. This method is independent of the artistic filter itself: the filter changes the crop, while the compositor controls where and how strongly the changed crop is placed in the original frame.

Foreground preservation is a related image-composition problem. A hand that passes in front of the face should remain visually above the transformed face. Hand landmarks can be used to estimate a region that is excluded from the stylized layer, allowing the original hand pixels to remain visible. This approach fits the project's gesture interface, although a landmark-derived region is only an approximation of the full hand silhouette.

4. Web Application Technologies

FastAPI is used to provide the local web application and HTTP endpoints [5]. Uvicorn runs the application server, while Jinja2 renders the main page. The browser-side interface is implemented with HTML, CSS and JavaScript. MediaPipe Tasks Vision is loaded in the browser for the main live perception workflow. OpenCV, NumPy and Pillow support Python-side image operations and supporting modules.

OpenCV and NumPy are relevant as supporting image-processing tools. OpenCV provides operations for image resizing, colour conversion, image encoding and other frame-level processing, while NumPy provides array operations for image channels and masks. These tools do not by themselves identify a face or interpret a gesture; they provide the lower-level operations used around the machine-learning outputs. The project combines these general-purpose operations with MediaPipe perception tasks and application-specific control logic.

FastAPI, Uvicorn and Jinja2 provide the web application structure around the live interface. The browser presents the camera view and controls, while the Python application serves the page and exposes supporting routes for style metadata, frame

processing and capture management. This separation is useful because the browser can perform the main live perception workflow while the server handles application delivery and supporting operations. The repository also contains Python helper modules and a separate desktop entry point, which should be distinguished from the browser's primary live path.

5. Comparative Synthesis and Research Gap

Prior tools address individual parts of the problem: hand-landmark models locate key points, segmentation models estimate pixel categories, image-processing libraries manipulate frames, and web frameworks expose application services. These components do not automatically provide a complete portrait-stylization experience. An application must still define how gestures are interpreted, how the selected style is controlled, how the head mask is aligned with the camera frame, how the foreground hand is preserved and how snapshots are stored.

The project's focus is this integration. It combines gesture-driven style navigation, intensity adjustment, pinch-based capture, head-region compositing and manual fallback controls in one interface. The available evidence supports a description of the architecture and visible behaviour, but it does not establish that this integration is more accurate or faster than alternative systems. A comparative performance claim would require a common test set, measured response times and a defined baseline.

6. Position of the Present Work

The project does not claim to improve the underlying MediaPipe models. Instead, it demonstrates how hand landmarks, segmentation masks and image effects can be integrated into an interactive application. The design emphasis is on the interaction mapping, region-specific transformation and the preservation of foreground objects. This distinction is important when interpreting the results: the screenshots demonstrate application behaviour, not a new model benchmark.

Table I. Summary Of Reviewed Works

Ref.	Approach	Data / Input	Main limitation
[1] 2019	MediaPipe perception-pipeline framework	General-purpose vision tasks	Framework requires task-specific models and application logic.
[2] n.d.	Hand Landmarker; 21 hand key points	Hand images / landmark output	Landmarks describe pose; temporal rules are needed to recognise swipes.
[3] n.d.	Selfie multiclass image segmentation	Selfie images; pixel confidence maps	Segmentation output may be uncertain around hair, accessories and occlusion.
[4] 2000	OpenCV image-processing library	Image arrays and video frames	Provides processing primitives, not semantic face or hand understanding.
[5] n.d.	FastAPI web framework	HTTP requests and image payloads	Provides API infrastructure, not computer-vision inference.
[6] 2026	FLUX.2 [klein] image-editing model	Prompt and image crop	Generative path needs compatible GPU or hosted service; not guaranteed real-time.
[8] 2026	Face Gesture Stylizer integration	Project source and four supplied screenshots	Evidence is qualitative; no labelled accuracy or controlled benchmark dataset.

III. Proposed System And Methodology

1. System Architecture

The browser application follows a modular processing pipeline. The camera provides the current frame. A main canvas displays the live image, while a reduced detection canvas is used for visual inference. The hand landmarker produces hand coordinates, and the segmentation task produces pixel-level confidence maps. The gesture engine interprets movement, the style engine creates the selected artistic effect, and the compositor places that effect within the estimated head region.

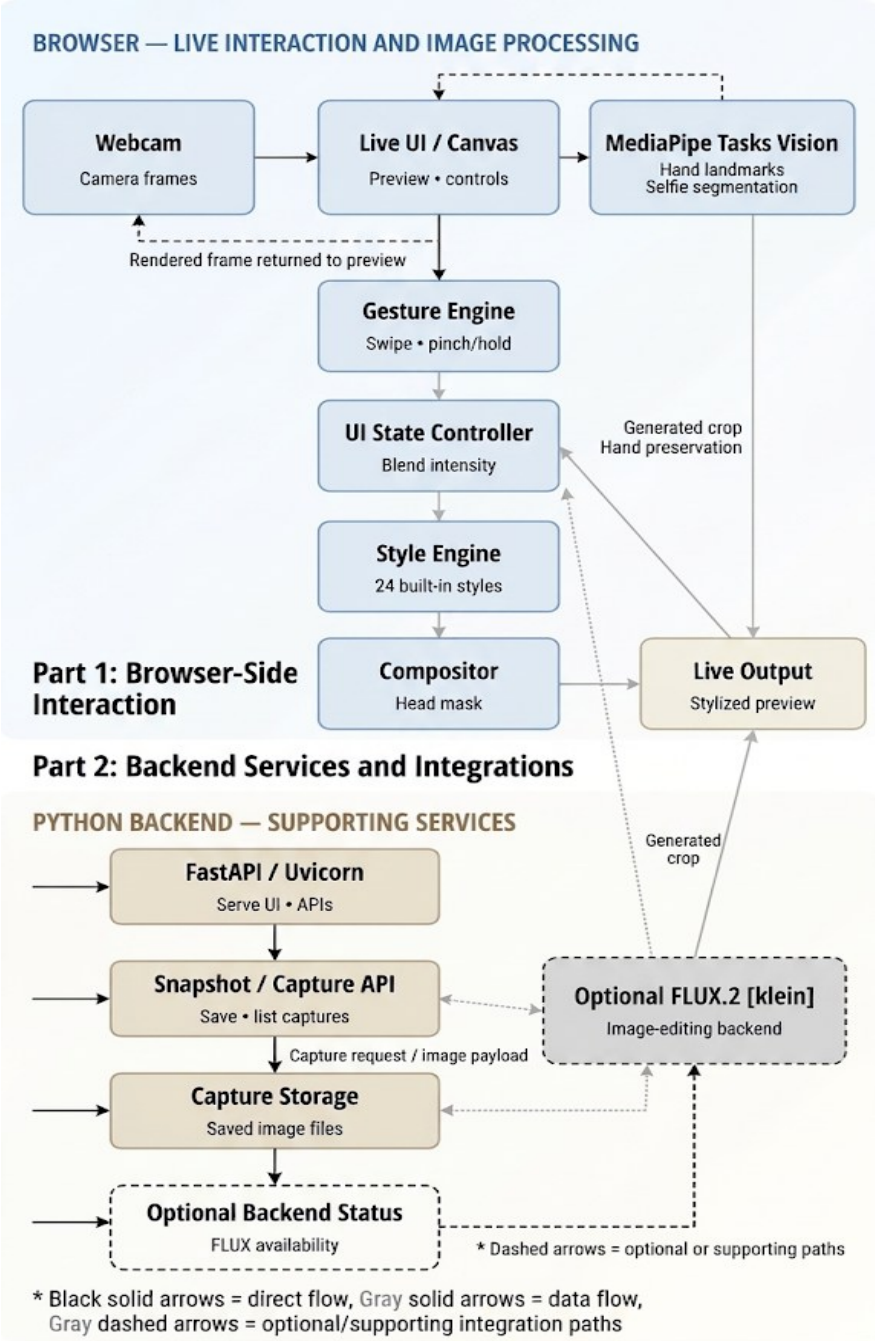


Fig. 5. System architecture of Face Gesture Stylizer showing the browser-side perception and rendering path, supporting FastAPI services, and optional FLUX.2 [klein] integration.

The architecture is divided into a browser-side real-time path and a Python backend path. The browser acquires the webcam frame, runs MediaPipe Tasks Vision, interprets gestures, updates style state and renders the selected effect through the estimated head mask. The FastAPI application serves the interface and supports API operations such as style metadata and snapshot management. The optional FLUX.2 [klein] path is shown separately because it requires a compatible configured backend and is not necessary for the built-in style pipeline.

TABLE II. MAIN SYSTEM COMPONENTS

Component	Function
Camera and interface	Acquire webcam frames and display the preview, controls and style carousel.
Vision pipeline	Run hand landmark detection and selfie segmentation.
Gesture engine	Track landmark movement and emit swipe or pinch events.
Style engine	Apply the selected built-in artistic transformation.
Face compositor	Blend the transformed head crop through the mask and preserve the hand region.
FastAPI server	Serve the interface, style metadata, capture storage and optional processing APIs.

2. Processing Workflow

At each update, the browser obtains a webcam frame and draws it to the display canvas. A smaller image is prepared for the vision models. Hand landmarks and segmentation output are produced, and the gesture engine updates its short history of palm positions. If a valid event is detected, the controller changes the selected style or intensity. The style engine processes the head crop, and the compositor blends the result with the original frame. A pinch-and-hold event requests a still-image capture.

- Acquire a webcam frame after camera permission is granted.
- Prepare a reduced detection image for the perception models.
- Run hand landmark detection and segmentation.
- Update the gesture history and classify any deliberate movement.
- Change the style index or blend intensity when a gesture event is valid.
- Apply the selected style to the estimated head crop.
- Composite the stylized crop through the soft head mask and preserve the hand.
- Render the final frame and save a still image when capture is requested.

3. Landmark Representation and Motion

The gesture engine uses the 21 hand landmarks to estimate a palm reference point. Rather than tracking a single fingertip, a palm centre is more stable because individual fingers can move independently. A hand-scale estimate based on the wrist and a middle-finger knuckle allows movement and pinch thresholds to be interpreted relative to the apparent size of the hand. This reduces sensitivity to the distance between the user and the camera.

A swipe is identified by comparing palm positions at different timestamps. Horizontal displacement is considered for left and right gestures, while vertical displacement is considered for upward and downward gestures. Minimum travel and speed thresholds reject small movements. Directional consistency helps distinguish a deliberate swipe from random motion, and a cooldown period prevents the same physical movement from triggering several commands.

4. Pinch-and-Hold

The pinch state is based on the distance between the thumb tip and index-finger tip, normalised by hand scale. A start threshold marks the beginning of a pinch, while a separate release threshold provides hysteresis. The pinch must remain stable for a short hold period before capture is requested. If the palm drifts too far or the fingertips separate, the hold is cancelled. This approach makes the capture command more intentional than a one-frame fingertip contact.

International Journal of Computational Research in Science and Technology
E-ISSN: 3108-0871, Volume 2 Issue 5
Sep-Oct-2026, PP 01-23

5. Gesture Mapping

Table III. Gesture-To-Command Mapping

Gesture	Command	Interface result
Open-hand swipe right	Next style	Select the following style card.
Open-hand swipe left	Previous style	Select the preceding style card.
Swipe upward	Increase intensity	Raise the blend-strength value.
Swipe downward	Decrease intensity	Lower the blend-strength value.
Thumb and index finger held together	Capture image	Save a still image.

IV. Head Segmentation And Image Compositing

1. Head Mask Generation

The browser pipeline uses MediaPipe selfie multiclass segmentation. Its confidence maps distinguish hair, face-skin, body-skin, clothes and background. The application combines the hair and face-skin channels to estimate the region for stylization. This is more suitable than using all skin pixels because body-skin may include the neck and hands. The mask is resized to the display-frame dimensions so that it remains aligned with the original camera image.

The mask boundary is softened to allow partial transparency near uncertain edges. The non-zero mask area can be used to determine a bounding region for the head crop. Processing a crop rather than the entire frame reduces unnecessary work and keeps the filter focused on the intended subject. The crop and mask must use consistent coordinate conventions; errors in resizing or mirroring can cause the effect to drift away from the face.

2. Fallback Behaviour

The vision pipeline includes a face-landmark fallback for situations in which the multiclass segmentation model is unavailable. The fallback estimates a face oval and an expanded upper region for hair. It can maintain a basic stylization path, but it is less semantically detailed than the segmentation model. Boundaries around ears, hats, glasses, hair accessories and the jaw may be less accurate.

3. Alpha Compositing

Let I(x,y) be the original camera frame, S(x,y) the stylized image mapped to the same coordinates, M(x,y) a soft head mask between zero and one, and α the user-selected intensity. The composite image is expressed as:

$$O(x,y) = [1 - \alpha M(x,y)]\, I(x,y) + \alpha M(x,y)\, S(x,y)$$

When α is zero, the original image is retained. When α is one, the stylized image is applied wherever the mask is active. Intermediate values produce a hybrid result. The interface shows the blend intensity as a percentage and allows the same state to be changed through the slider or vertical hand gestures.

4. Foreground Hand Preservation

A hand may pass in front of the face while the user performs a gesture. If the stylized layer is drawn over the hand, the visual depth order becomes incorrect. The compositor therefore removes an approximate hand-shaped region from the stylized layer using the hand landmarks. The original hand pixels remain visible. The quality of the cut-out depends on hand detection and the size of the region estimated from the landmarks.

5. Factors Affecting Mask Quality

Segmentation may be affected by low illumination, camera exposure, head pose, motion blur, hair texture and background similarity. Hats and glasses can create uncertain boundaries. Strong posterised or neon effects can make small mask errors more noticeable. The mask is therefore an estimated region rather than an exact facial boundary. A practical implementation should provide a mask-debug view and should avoid presenting uncertain segmentation as perfect.

V. Artistic Style Engine

1. Built-In Style Collection

The interface exposes 24 built-in styles. The style carousel presents a numbered card, a style name and a category. The currently selected style is also shown in the upper status area. The style engine receives the selected identifier and applies the corresponding image transformation to the head crop. The compositor then controls where the result appears in the final frame.

TABLE IV. EXAMPLES FROM THE STYLE CATALOGUE

Style name visible in the application	Visual direction
Pop Art Comic	High-contrast comic-book colour regions and graphic appearance.
Van Gogh Swirl	Painterly, impressionist-inspired treatment.
Cyberpunk Neon	Futuristic palette with neon colour emphasis.
Anime & Manga	Animation-inspired facial appearance.
Urban Graffiti	Vivid street-art palette and strong contrast.
Dark Gothic Noir	Dark, dramatic monochrome-inspired treatment.
Synthwave 80s	Retro-futuristic colour treatment.
Watercolor Dream	Soft fine-art and watercolor-inspired appearance.
Digital Glitch	Digital distortion and science-fiction styling.
Thermal Infrared	Scientific heat-map-inspired palette.

The four supplied screenshots directly show Pop Art Comic, Anime & Manga and Urban Graffiti. The carousel also displays names such as Van Gogh Swirl, Cyberpunk Neon, Dark Gothic Noir, Synthwave 80s, Watercolor Dream and Digital Glitch. The full catalogue count is 24 according to the interface and the style metadata endpoint.

2. Intensity and Style Navigation

The blend-strength control ranges from a natural view to a full transformation. The slider displays the selected percentage, and vertical gestures adjust the same application state. Horizontal gestures change the selected style. The interface uses a wipe transition when the style changes, providing visual feedback that a new effect has been selected. Manual carousel navigation remains available as an alternative.

3. Optional FLUX.2 [klein] Mode

The optional generative mode is separate from the built-in style engine. It sends a head crop to a FLUX.2 [klein] image-editing backend with a style prompt and blends the returned image through the live face mask. The repository describes a local Diffusers backend for a compatible NVIDIA GPU and a hosted API option when credentials are configured. The README states that live generative paths were exercised with mocks in the development environment. Thus, this report documents the integration path but does not claim that live FLUX inference was demonstrated by the screenshots.

Keeping the built-in effects independent of the generative path is important for availability. The application can continue to use local filters when a GPU server is not present, network access is unavailable or API credentials have not been configured. The generative path should be treated as an optional enhancement rather than a requirement for the main live interface.

VI. Implementation

1. Frontend Modules

The JavaScript controller coordinates the video element, main canvas, status indicators, style counter, FPS display, buttons, intensity slider, snapshot control, gallery and optional FLUX controls. It stores the current style index, previous style index, blend intensity and view state. During startup it loads style metadata, initialises the vision pipeline, binds user-interface events and requests camera access.

Table V. Principal Project Modules

Source file / module	Responsibility
static/js/app.js	Coordinates camera, UI state, render loop and control events.
static/js/vision_pipeline.js	Loads and runs browser hand and segmentation models.
static/js/gesture_engine.js	Recognises swipe and pinch/hold events from landmarks.
static/js/style_filters.js	Contains the browser-side artistic effects.
static/js/face_compositor.js	Applies the effect through the head mask and hand cut-out.
core/face_segmenter.py	Python-side segmentation and mask helpers.
core/gesture_detector.py	Python-side gesture detection and fallback logic.
core/style_engine.py	Python-side style definitions and image effects.
core/flux_client.py	Optional local or hosted FLUX integration.

2. Backend and API

The Python web application uses FastAPI and Uvicorn. The root route renders the main page through a Jinja2 template. Static assets are served from the static directory, and saved images are made available from the captures directory. The /api/styles route returns the available style metadata and count. Additional endpoints support frame processing, snapshot saving, capture listing and optional FLUX status or editing.

Table VI. Main Web Api Endpoints

Endpoint	Method	Purpose
/	GET	Render the main web interface.
/api/styles	GET	Return the style list and count.
/api/process_frame	POST	Accept an image and return processed output and gesture metadata.
/api/save_snapshot	POST	Save a submitted image in the captures directory.
/api/captures	GET	List saved images and basic metadata.
/api/flux_status	GET	Report whether the optional FLUX backend is available.
/api/flux_face	POST	Submit a head crop for optional generative editing.

3. Image Data Handling

Image payloads can be provided as base64 data URLs or base64 image strings. The server decodes the payload into an OpenCV image array and checks whether decoding succeeded. The snapshot route creates a filename using the selected style and a timestamp. The capture-list route returns filenames, URLs, file sizes and creation times. Invalid image data is handled through an error response rather than being passed silently into the processing pipeline.

4. Development Environment

The project requirements include FastAPI, Uvicorn, Jinja2, Pydantic, NumPy, OpenCV, Pillow and Requests. MediaPipe is included for desktop and helper modules. The browser's primary live vision workflow uses MediaPipe Tasks Vision in JavaScript, loaded from a CDN. The repository also includes app_cv.py as a separate OpenCV desktop path. The README notes that some Python helper modules expect legacy MediaPipe Solutions interfaces and may use weaker fallback behaviour with newer versions.

5. Execution Procedure

- Extract the supplied project archive into a local folder.
- Open a terminal in the project directory and install the requirements.
- Run the web application using python app_web.py or the supplied launcher.
- Open http://localhost:8000 in a modern browser.
- Grant webcam permission and wait for the browser vision models to initialise.
- Use gestures or manual controls to browse styles, adjust intensity and capture images.

Camera access is restricted by browser security rules. Localhost is treated as a development exception in common browsers, while a deployed application should use HTTPS. The first model load requires internet connectivity to retrieve the runtime and model assets from the configured hosts.

VI-A. Detailed Design And Operational Considerations

1. Camera Acquisition and Frame Scheduling

The browser obtains camera frames through the standard media-device interface after the user grants permission. The video element acts as the live source, while canvas elements are used for drawing and processing. The main display canvas is responsible for presenting the current frame and the final stylized output. A separate, smaller detection canvas allows the perception models to operate on a reduced image. This separation is useful because the display resolution and the resolution required for inference do not need to be identical.

A live application must balance visual quality and responsiveness. Processing every frame at full camera resolution can increase inference and drawing cost. Reducing the detection image can lower the amount of data processed by the models, although very small images may remove details needed for hand landmarks or segmentation boundaries. The appropriate size is therefore a practical trade-off between detail and speed. The supplied screenshots show that the application exposes a frame-rate indicator, but they do not provide enough information to infer the internal frame-scheduling policy or the exact time spent in each stage.

The application should also avoid allowing slow operations to accumulate. If one inference request is still running, a second request should not create an unbounded queue of stale frames. A well-behaved real-time loop processes the most recent available frame and discards frames that are no longer useful for the current preview. This is a general design consideration for interactive video pipelines; the project's visible frame-rate indicator is a useful user-facing signal, but a detailed profiler would be needed to confirm the timing of individual operations.

2. Gesture State Management

Gesture recognition is more reliable when it is represented as a small state machine rather than a collection of unrelated distance checks. For swipe recognition, the engine can be understood as moving through tracking, candidate movement, accepted gesture and cooldown states. While tracking, it updates the palm-centre history. When the displacement exceeds the relevant movement condition, the engine evaluates direction and consistency. If the movement satisfies the gesture conditions, it emits one event and enters a cooldown state. After the cooldown, it resumes normal tracking.

This state-based approach helps avoid duplicate commands. For example, when a user waves the hand to the right, the hand may naturally move back toward its starting position. Without a cooldown or directional rule, the return movement could be interpreted as a second swipe in the opposite direction. A short motion history also prevents a single noisy landmark estimate from immediately changing the style. The system should consider both the amount of movement and the time over which it occurred.

Pinch capture has a different state sequence. The engine first detects that the thumb and index finger have approached one another. It then tracks whether the pinch remains stable for the required hold interval. If the fingers separate or the palm moves too far, the candidate capture is cancelled. If the hold condition is satisfied, the engine emits a capture event and applies a cooldown to prevent repeated snapshots while the fingers remain together. Separate start and release thresholds reduce rapid switching when fingertip distance fluctuates near a boundary.

The project's gesture panel communicates the intended controls to the user: right and left swipes change styles, up and down swipes adjust intensity, and pinch-and-hold saves a photo. This visible mapping is important because gesture interfaces are not self-explanatory. A person should be able to learn the available commands without guessing. The application also retains manual controls so that a user can recover from a missed gesture or operate the interface when the hand is outside the camera view.

3. Coordinate Systems and Image Alignment

The camera image, hand landmarks, segmentation map and head crop may have different dimensions. Correct alignment is therefore a central implementation requirement. A landmark coordinate expressed relative to the detection image must be mapped consistently to the display image before it is used to draw a hand cut-out. Similarly, the segmentation mask must be resized using the same orientation and mirroring convention as the displayed camera frame. If the video is mirrored for a natural selfie view but the mask is not mirrored, the stylized region may appear displaced.

Cropping introduces another coordinate transformation. The bounding rectangle around the head region is expressed in frame coordinates, while the style filter operates on the cropped image. The compositor must map the processed crop back to the same location in the original frame. The mask must be cropped using the same rectangle and resized to the exact output dimensions. A mismatch of even a few pixels can create a visible halo, a cut edge or an effect that does not follow the face.

Interpolation also matters. Nearest-neighbour resizing preserves hard labels but can create jagged boundaries. Bilinear interpolation produces smoother values and is suitable for a soft confidence mask, although it may blur the transition. The mask can be feathered to reduce visible edges. These choices should be evaluated visually because the best interpolation depends on the resolution of the segmentation output and the strength of the artistic effect.

4. Style Rendering and Compositing Responsibilities

The style engine and compositor have different responsibilities. The style engine changes the appearance of an input crop by applying a visual transformation. The compositor decides where that transformed crop should appear in the final frame. Keeping these responsibilities separate makes the system easier to extend: a new style can be added without rewriting the head-mask logic, and an improved segmentation method can be adopted without changing every style filter.

A style transformation may change colour, contrast, texture or local image structure. A comic treatment may use a limited palette and high contrast, while a watercolor-like treatment may soften edges and produce smoother colour transitions. A neon treatment may emphasise bright colours and dark regions. The exact visual character is determined by the filter implementation and its parameters. The project interface identifies styles by name and category, allowing the user to browse them without needing to understand the image-processing operations behind each one.

The blend-intensity value provides a common control over the result. At a low value, the original crop remains dominant; at a high value, the selected style becomes more prominent. Applying the same intensity parameter across the style engine and

compositor gives the user a consistent control even though different filters may have different visual characteristics. The screenshots show the slider at its maximum value, so they document the full-transform setting rather than a comparison across multiple intensity levels.

5. API Responsibilities and Separation of Processing Paths

The FastAPI application provides the web page and routes for supporting operations. The style route returns the catalogue used by the interface. The capture route saves an image submitted by the client, and the capture-list route returns information about saved files. The frame-processing route accepts image data and returns a structured response. The optional FLUX routes report backend availability and accept a crop for generative editing. This API organisation separates user-interface responsibilities from storage and optional server-side processing.

The browser-based live workflow and the Python-side processing modules should be described accurately as related but distinct paths. In the browser workflow, MediaPipe Tasks Vision performs the main hand tracking and segmentation in JavaScript. The Python server serves the application and exposes supporting endpoints. The repository also contains Python image-processing modules and a desktop OpenCV entry point. A report that treats all these components as one single inference path would obscure the actual software structure.

Error handling is necessary because camera and model operations can fail independently. The user may deny camera permission, a model asset may not load, an image payload may be invalid, or an optional generative backend may be unavailable. A useful interface should expose a clear status message and preserve manual operation where possible. The FLUX path should not be allowed to prevent the built-in filter workflow from operating.

6. Privacy and Responsible Use

A webcam application processes personal visual information. Camera permission should be requested explicitly, and the interface should make it clear when the camera is active. A still image should be saved only after a deliberate capture action. If the application is deployed for multiple users, the capture directory should have appropriate access controls and a defined retention policy. Users should be able to remove saved images when they are no longer needed.

The optional generative editing path has an additional data-flow consideration. If a head crop is sent to a hosted service, the user should be informed that image data leaves the local device and is processed by an external provider. The application should explain the service configuration and applicable data-handling terms. The artistic output should not be represented as a faithful biometric image or as evidence of a person’s real appearance.

7. Evaluation Protocol and Metrics

A complete evaluation should be organised around separate questions for gesture recognition, segmentation, compositing, performance and usability. Separating these questions prevents a visually successful screenshot from being treated as proof of every subsystem. Each experiment should identify the device, browser version, camera resolution, lighting conditions, model configuration and number of trials. This information is necessary for another developer to repeat the evaluation.

For gesture recognition, test recordings should contain labelled examples of left swipe, right swipe, upward swipe, downward swipe, pinch-and-hold and ordinary non-gesture movement. The set should include different movement speeds, short and long gestures, diagonal motion, hand jitter, partial occlusion and the return movement after a swipe. Precision measures how many emitted events are correct, while recall measures how many intended gestures are detected. False triggers per minute measure the frequency of unintended commands during ordinary movement.

For segmentation, the predicted head mask should be compared with a manually annotated reference mask. Intersection over Union measures the overlap between the predicted and reference regions relative to their union. Dice similarity measures the overlap relative to the total size of both masks. Results should be reported across different lighting conditions, head poses, hairstyles and backgrounds because a single controlled scene is not representative of everyday camera use.

For compositing, a fixed source image and a known synthetic mask can be used to verify that pixels outside the mask remain unchanged. A separate test can place a synthetic hand region over the face and check whether the hand remains visible after compositing. Performance evaluation should record average frames per second, median frame time and high-percentile frame time over a fixed duration. User evaluation can measure task completion time, accidental activation, retries and user-reported ease of learning.

No such complete numerical evaluation is claimed in the present report. The supplied screenshots provide qualitative evidence of the running interface and selected effects. The repository includes test files for the gesture engine, vision pipeline, compositor, style rendering and application smoke path, but the presence of these tests is not the same as a documented fresh test run. This distinction keeps the reported findings aligned with the available evidence.

8. Interaction Design and User Feedback

The interface is arranged around the live camera preview. The preview occupies the largest area because the user needs to see the effect while interacting with it. A side panel describes the available gestures, and the style carousel provides a second way to browse the collection. The selected style is repeated in the header so that the current state remains visible even when the user is looking at the camera preview rather than the carousel.

The gesture instructions serve as an embedded guide. They describe the direction of the hand movement and the action that follows. This reduces the learning effort associated with a gesture-driven interface. The on-screen notification shown in one of the supplied captures is another feedback mechanism: it confirms that a swipe was interpreted and identifies the current style. Without such feedback, a user may not know whether the application missed the gesture or changed to an unexpected style.

The intensity slider is important because a full transformation may not always be preferred. The displayed percentage gives the user a clear indication of the current setting. The slider and gesture controls should update the same underlying intensity state so that the visual output remains consistent regardless of the input method. The split-view and mask controls also help users inspect the difference between the original camera image and the processed output.

9. Failure Scenarios and Recovery

A real-time camera application must handle failure at several stages. Camera access may be denied, the camera may already be in use by another application, or the video stream may stop. The vision runtime may fail to load because of a network problem, and a model may not initialise correctly. During operation, the hand may leave the frame or become partially occluded. Segmentation may return an uncertain mask, and a server endpoint may receive invalid image data.

A robust interface should distinguish these conditions rather than displaying one generic error. Camera failure should be reported separately from model-loading failure. A missing hand should not necessarily stop face stylization, and a missing segmentation mask should not be mistaken for a successful high-confidence mask. When a gesture is not detected, the user should still be able to use the carousel or manual buttons. When the optional generative backend is unavailable, the built-in filter collection should remain available.

Recovery can be gradual. The application can continue showing the original camera frame while a model is loading, then enable the corresponding feature after successful initialisation. If hand tracking is temporarily lost, the gesture engine can clear or reset its motion history so that the next detected hand does not inherit an old palm position. If the segmentation output becomes invalid, the compositor should avoid applying a stale mask to a new frame. These are general reliability requirements for the described architecture.

10. Design Trade-Offs

The project balances visual quality, computational cost and interaction responsiveness. A high-resolution camera image contains more detail but increases the cost of inference and rendering. A smaller detection image can improve speed but may

make fine boundaries harder to estimate. A complex artistic effect may produce a more distinctive result but require more image operations. A softer mask can reduce visible edges but may also blend the effect into nearby regions. These trade-offs should be evaluated on the target hardware rather than assumed to have one universally optimal setting.

The system also balances automatic and manual interaction. Gesture control is convenient when the user is positioned in front of the camera and the hand is clearly visible. Manual controls are more predictable when the hand is outside the frame or when the user prefers conventional input. Retaining both methods makes the application more accessible and reduces dependence on a single perception component.

11. Reproducibility and Reporting

A reproducible project report should identify the source version, runtime environment and steps used to start the application. It should distinguish the browser-based pipeline from the Python desktop path, and it should state whether optional services were configured. Screenshots should be accompanied by captions describing the selected style and visible controls. Performance observations should identify the measurement method and duration. If a result is based only on a screenshot, it should be described as a visual observation rather than a measured benchmark.

For this project, the available screenshots show the application at localhost:8000 and document the selected styles and interface controls. The displayed frame-rate values are momentary readings. The project archive contains test files, but no test execution log or formal accuracy dataset is included in the supplied evidence. Accordingly, the present report documents the implemented design and visible behaviour and proposes a repeatable evaluation method without inventing unobserved measurements.

12. Practical Applications

The main use of Face Gesture Stylizer is creative portrait transformation. The same design principles can be applied to interactive photo booths, virtual event experiences, educational demonstrations of computer vision and hands-free media interfaces. The gesture-to-command mapping is intentionally small and memorable, which makes it suitable for a short demonstration. The region-specific compositor also illustrates how semantic segmentation can be combined with conventional image processing to produce a more controlled visual effect.

The project can also serve as a teaching example. It demonstrates the difference between landmark detection and segmentation: landmarks describe key points on a hand, while segmentation estimates the category of each pixel. It demonstrates the difference between a style filter and a compositor: the filter changes appearance, while the compositor controls where that appearance is applied. Finally, it illustrates the importance of separating the browser's real-time perception workflow from server-side routes and optional generative services.

13. Security and Deployment Considerations

The current project is demonstrated on localhost. A public deployment would need HTTPS for camera access, careful handling of uploaded image data, validation of image dimensions and file types, and access controls around saved captures. The capture endpoint should avoid accepting unbounded or malformed payloads. The application should also consider rate limits if the processing routes are exposed to multiple users. These measures are deployment considerations and are not claims that a public production deployment has already been completed.

If hosted generative editing is enabled, the application should make the external transfer explicit and should avoid sending more image data than required. A head crop is preferable to an entire camera frame when the generative task only needs the face region. The user should understand whether the request is processed locally or sent to a hosted service. This distinction is especially important for a webcam application because the input may contain identifiable personal imagery.

14. Comparison with Conventional Filter Interfaces

A conventional photo-filter interface typically applies a selected effect to the entire image or asks the user to manually define a region of interest. Whole-frame processing is simple to implement, but it changes the background and other objects along with the subject. Manual region selection gives the user more control, but it can be inconvenient in a live camera application because the region must be adjusted as the person moves. Face Gesture Stylizer instead estimates the head region continuously and uses that estimate to guide compositing. This reduces the need for manual selection, although it introduces dependence on segmentation quality.

The project also differs from a standard button-driven filter gallery through its gesture mapping. The user can move to the next or previous style with a horizontal swipe and adjust the effect strength with a vertical movement. A pinch-and-hold gesture is assigned to image capture. The same operations remain available through visible interface controls, so the gesture method is an additional interaction channel rather than the only method of operating the application.

The region-specific approach has its own limitations. If the mask is inaccurate, part of the face may remain unprocessed or nearby background pixels may be affected. If a hand is not detected, the compositor may not preserve it correctly when it crosses the face. A conventional full-frame filter does not have these particular mask-boundary problems, although it also cannot preserve the natural background in the same way. The design therefore trades the simplicity of whole-frame effects for a more focused portrait result that depends on real-time perception.

The intended benefit is best understood as a combined interaction and rendering experience: the user can select an artistic style without repeatedly reaching for a control, and the transformation is constrained to a semantically meaningful region. Whether this approach is faster or more accurate than manual selection would require a controlled user study. The current screenshots establish the visual behaviour of the application but do not provide comparative task-time or preference data.

15. Summary of the Methodological Pipeline

The complete method can be summarised as a sequence of perception, decision and rendering operations. Perception begins with a camera frame and produces two forms of information: hand landmarks and semantic segmentation. The gesture engine uses landmark history to decide whether a command has occurred. The application controller updates the style or intensity state, and the style engine transforms the head crop. The compositor then uses the mask and hand region to combine the transformed crop with the original image. The final frame is displayed, and a still image is saved only when the capture action is requested.

This pipeline clarifies the role of each subsystem. The hand model does not directly stylize the face; it provides coordinates for interaction and occlusion handling. The segmentation model does not choose the artistic style; it estimates the pixels that should receive the transformation. The style engine does not decide where the effect belongs; it produces the transformed image. The compositor combines these outputs into the final display. This separation is the central architectural principle of the project and provides a clear basis for future testing and improvement.

16. Data Flow and State Synchronisation

The application maintains several kinds of state at the same time. The camera state indicates whether a video stream is available. The vision state indicates whether the hand and segmentation models have been initialised. The interaction state stores the selected style, previous style, blend intensity and any active gesture cooldown. The rendering state contains the latest frame, head crop, segmentation mask and final composite. Keeping these values separate helps prevent one subsystem from accidentally overwriting information owned by another subsystem.

A gesture event should update the application state through a single controller path. For example, a right swipe changes the selected style index, after which the style engine receives the new style identifier. A vertical gesture changes the intensity value, after which the compositor receives the updated blend strength. A pinch-and-hold requests a capture using the latest

completed frame. This event-driven arrangement avoids placing interface-specific actions inside the landmark detector itself. The detector identifies a gesture; the controller decides what that gesture means for the application.

The selected style and intensity should also remain synchronised between the visual controls and the gesture engine. If the user moves the slider, the value used by the compositor should change immediately. If the user adjusts intensity with a hand gesture, the slider should reflect the new value. The same principle applies to style selection: a carousel click and a swipe should update the same style index. Sharing a single state representation avoids inconsistent displays in which the selected card and rendered effect do not match.

Capture operations require similar coordination. The pinch detector should emit a capture event only after its hold condition is satisfied. The controller can then request a snapshot using the current frame and selected style metadata. The snapshot endpoint stores the submitted image, while the capture-list endpoint provides information for the gallery. The interface should make it clear whether the capture has been saved and should avoid treating a gesture event as proof that a file was successfully written.

17. Model Loading and Runtime Availability

The browser vision pipeline depends on the successful loading of the MediaPipe runtime and its model assets. The first launch may therefore take longer than later use, particularly when the browser must download model files. A clear loading state is useful because a blank or unchanged preview may otherwise be mistaken for a frozen application. The interface should distinguish camera readiness from model readiness: the camera can be active even when hand tracking or segmentation has not completed initialisation.

The hand and segmentation tasks are logically independent. If hand tracking is unavailable, the user may still be able to use manual style controls and may still receive a face mask if segmentation is working. If segmentation is unavailable, the application may use its face-landmark fallback or temporarily display the unmodified camera frame. These degraded modes are preferable to applying an old or invalid mask to a new image. A model status indicator can help the user understand which capabilities are active.

The project uses browser-side Tasks Vision for its main live perception path, while the Python repository includes supporting processing modules and a desktop entry point. The two paths have different dependency and compatibility considerations. The README notes that some Python helper code expects legacy MediaPipe Solutions interfaces, whereas the browser path uses the Tasks Vision API. Installation and troubleshooting instructions should identify which path is being used so that a compatibility issue in one path is not incorrectly attributed to the other.

18. Quality Assurance for Visual Output

Visual quality should be checked at several points in the pipeline. The raw segmentation mask should be inspected before the effect is applied, because an inaccurate mask is difficult to correct at the final compositing stage. The cropped head region should be checked for clipping, scaling and orientation. The stylized crop should be checked for valid dimensions and colour channels. Finally, the composite should be inspected around the hairline, ears, jaw and any hand that crosses the face.

A useful manual test sequence begins with a neutral camera scene and a stationary head. The user selects several styles and compares the transformed region with the original frame. The user then moves the head slightly, changes distance from the camera and turns to different angles. Next, the user places a hand in front of the face and performs each gesture. The test should observe whether the mask follows the head, whether the hand remains in front, whether style changes occur once per swipe and whether the capture command requires a deliberate hold.

The same sequence should be repeated under different lighting and backgrounds. Bright light can improve image detail but may cause overexposure; low light can increase noise and blur. A background with colours similar to hair or skin may make

segmentation more difficult. Accessories such as hats and glasses may change the apparent boundary of the head. These tests do not replace quantitative evaluation, but they can reveal failure cases that should be included in a formal dataset.

19. Limitations of the Available Evidence

The available project materials include source code, a README and four screenshots. They support a description of the architecture, the available interface controls and the styles visibly demonstrated. They do not include a labelled dataset of hand gestures, manually annotated head masks, a benchmark report or a user-study dataset. It is therefore not possible to calculate gesture precision, segmentation overlap, false-trigger rate or comparative usability from the supplied evidence alone.

The screenshots show frame-rate values at individual moments. They do not specify the duration over which those values were observed, the hardware model, the browser version or the camera resolution. They should be cited as visual observations only. Likewise, the presence of a FLUX.2 control establishes that the interface contains an optional generative mode, but does not establish that a live external or local generation completed successfully during the captured session. The report maintains these distinctions to avoid overstating the results.

20. Functional Requirements and Acceptance Conditions

The main functional requirements can be expressed in terms of observable behaviour. When the camera is active and a hand is detected, a deliberate right swipe should advance the selected style by one position. A left swipe should move to the preceding style. The selected style name and carousel highlight should update together. An upward or downward gesture should change the blend-intensity value, and the displayed slider should reflect the new value. A stable pinch-and-hold should request one capture rather than repeated captures while the fingers remain together.

The image-processing requirement is that the artistic effect should be limited to the estimated head region. Pixels outside the mask should remain close to the original camera frame, apart from interface overlays. When a hand is in front of the face, the hand should remain visible above the transformed region. The user should be able to change styles through the carousel even when gesture detection is unavailable. These acceptance conditions can be tested individually and then as an integrated scenario.

A complete acceptance test should include the following sequence: start the application, grant camera permission, wait for the vision models, select a style manually, change to another style using a right swipe, return using a left swipe, adjust intensity in both directions, place a hand in front of the face, and capture a still image using pinch-and-hold. The tester should confirm that the style indicator, carousel, intensity value and capture gallery remain consistent. Any failed condition should be recorded with the browser and device configuration.

20.1 Non-Functional Requirements

Responsiveness is important because the application is intended for live interaction. A noticeable delay between a gesture and its visual response can make the interface feel unreliable. The system should therefore keep the perception and rendering loop responsive and should avoid queuing old frames. Visual consistency is also important: the head mask should remain aligned with the face as the user moves, and the selected effect should not flicker unnecessarily between adjacent frames.

Usability requires clear gesture instructions, visible feedback and a manual fallback. Reliability requires that a temporary failure in one component does not unnecessarily disable all other features. Maintainability benefits from separating the controller, vision pipeline, gesture engine, style engine and compositor. Privacy requires that the user knows when the camera is active and when an image is saved or sent to an external service.

20.2 Acceptance Evidence in the Supplied Material

The supplied screenshots demonstrate that the interface can display the live preview, style carousel, gesture instructions, intensity slider and selected style indicator. They show three selected visual styles and a notification for a left swipe. These

images support the corresponding visible acceptance conditions. They do not show a complete sequence for every gesture or prove the accuracy of the detection logic across multiple attempts. Such claims would require a recorded test procedure and results.

21. Deployment and Operational Considerations

For local demonstration, the application is launched from the project directory and opened through localhost:8000. The browser asks for camera permission, and the browser-side vision assets are loaded from configured external locations. This arrangement is convenient for development because localhost is treated as a secure context by common browsers. A public deployment should use HTTPS and should document the camera permission requirement before the user starts the scanner.

The server should validate image payloads before decoding them. The image dimensions and encoded payload size should be bounded to avoid excessive memory use. Capture filenames should be generated by the application rather than accepted blindly from a client. If the capture directory is exposed through a static route, the deployment should consider who can access the saved images. These are standard operational safeguards for an application that accepts and stores webcam images.

The optional generative editing path introduces an additional dependency. A local GPU backend must have compatible model weights and runtime libraries. A hosted API requires credentials and network access. The application should report the backend status clearly and should not imply that generative editing is active when it is unavailable. The built-in filters should remain usable without this service so that the core demonstration does not depend on external generation.

21.1 Installation and First-Run Behaviour

A typical local installation consists of extracting the project archive, opening a terminal in the project folder, installing the packages listed in requirements.txt and starting the FastAPI application. The browser is then directed to the local address. On the first visit, the user may need to wait for the JavaScript runtime and model files to load. The camera prompt must be accepted before the live preview can begin. If the browser has previously blocked camera access, permission settings may need to be changed.

Troubleshooting should follow the dependency chain. If the page does not load, check that the server process is running and that the local port is available. If the page loads but the camera preview is blank, check browser permission and camera availability. If the camera works but hand gestures are not detected, check whether the hand model initialised and whether the hand is visible in the frame. If the face effect is missing, check the segmentation model status and whether the fallback path is active. This staged diagnosis is more efficient than treating every issue as a general application failure.

22.2 Maintainability and Extension

The modular design supports future extension. A new style can be added to the style catalogue and associated with its rendering function without changing the gesture engine. A new gesture can be added by extending the gesture state logic and mapping its event in the application controller. A more accurate segmentation model can replace the current mask provider as long as it returns a mask in the coordinate system expected by the compositor. This interface-based separation reduces the number of components that need to change for a feature update.

The same principle applies to the optional generative path. It can remain behind a separate client interface so that a local model or hosted service can be selected without changing the built-in filters. The controller should treat the returned generated crop as an input to the compositor, not as a replacement for the complete camera frame. This keeps the head mask and hand-preservation behaviour consistent between built-in and generative effects.

23. System Reliability And Failure Handling

A live camera system contains multiple components that can fail independently. Camera access, model loading, landmark detection, segmentation, image rendering, server communication and snapshot storage should therefore be treated as separate

stages. If the camera permission is denied, the application cannot obtain a live frame, but the page itself may still load. If the hand model is unavailable, the user may still browse styles manually. If segmentation fails, the system should avoid applying a stale mask and should either use the documented fallback or display the original frame until a valid mask is available.

The gesture engine should reset its temporal history when the hand disappears for a meaningful interval. Otherwise, a newly detected hand position may be compared with an old position and create a false swipe. A similar reset is useful when the camera stream restarts or the video dimensions change. The controller should also clear active cooldown and candidate states when the user stops the camera. These practices prevent state left over from an earlier session from affecting a later one.

The rendering pipeline should validate the dimensions and availability of each input before compositing. The style image, mask and head crop must be compatible in size and orientation. If a filter returns an invalid image, the original frame should remain visible rather than allowing the rendering loop to fail. If a capture request is received while a frame is not ready, the controller should either wait for the next valid frame or report that the capture could not be completed. This is preferable to reporting a successful save before the server has confirmed that the file was written.

Server-side routes also need predictable failure responses. Base64 decoding may fail, an image may be empty, or an optional processing backend may be unavailable. The API should return a controlled error response with a useful status rather than terminating the application. The FLUX status route allows the interface to distinguish an unavailable generative backend from the normal built-in style workflow. This separation is important because optional functionality should not become a single point of failure for the core application.

- **U.1 User Feedback During Failure**
  When a component is unavailable, the interface should communicate what remains usable. A message such as camera permission required is more useful than a generic processing error. A model-loading message should explain that the user may need to wait for initialisation. If hand detection is unavailable, the interface can direct the user to the carousel and manual controls. If generative editing is unavailable, the interface can continue to offer the built-in styles. Clear status messages reduce repeated gestures and unnecessary user frustration.
- **U.2 Reliability Testing**
  Reliability tests should deliberately interrupt the normal workflow. Examples include denying camera permission, disconnecting the network before model assets load, covering the hand, moving the face partially out of frame, submitting malformed image data and disabling the optional FLUX backend. The expected behaviour should be written before each test. A successful test is not simply that the application does not crash; it should also preserve a useful fallback and explain the problem to the user.

A long-duration test can be used to identify memory growth, stale state and frame-rate degradation. The application should be left running for a fixed period while the user changes styles and performs gestures. The test should record whether the preview remains active, whether gesture commands continue to work and whether the capture gallery remains accessible. Such a test would complement the short screenshot demonstrations, which show selected moments rather than long-term stability.

These reliability considerations are proposed verification procedures based on the architecture described in the project files. They should not be read as claims that every failure scenario has already been tested. The distinction between implemented behaviour, visible screenshot evidence and recommended validation remains important for an accurate academic report.

VII. Results And Discussion

1. Screenshot-Based Observations

The supplied screenshots show the application running at localhost:8000. The interface contains a large live camera preview, a selected-style indicator, a carousel of style cards, a hand-gesture instruction panel, a blend-intensity slider, a snapshot control

and a capture gallery. The style indicator shows a collection count of 24. The screenshots directly demonstrate three selected effects: Pop Art Comic, Anime & Manga and Urban Graffiti.

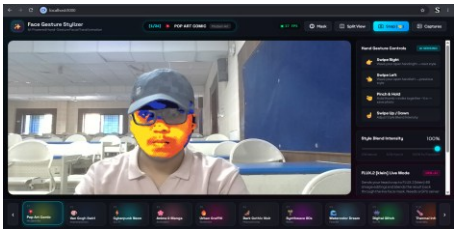


Fig. 1. Live application with Pop Art Comic selected and blend intensity at 100%.

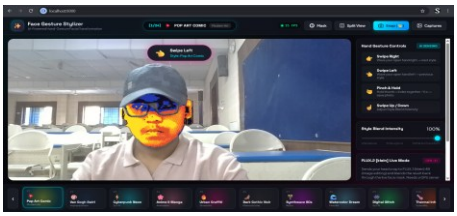


Fig. 2. Pop Art Comic output with an on-screen left-swipe notification.

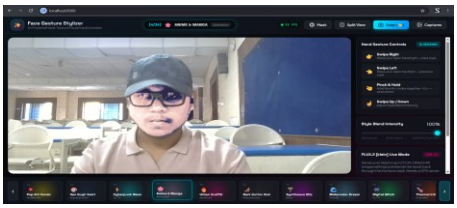


Fig. 3. Anime & Manga style selected in the live preview.

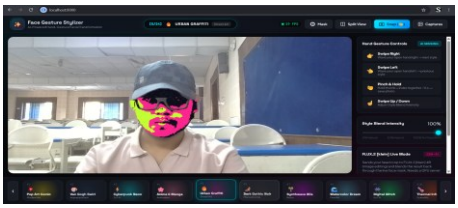


Fig. 4. Urban Graffiti style selected in the live preview.

2. Observed Interface Behaviour

Table VII. Observations From The Supplied Screenshots

Observed item	Evidence and interpretation
Style catalogue	The header shows 1/24 and the carousel displays named style cards.
Style switching	The supplied captures show Pop Art Comic, Anime & Manga and Urban Graffiti.
Gesture instructions	The side panel describes horizontal swipes, vertical intensity adjustment and pinch capture.
Gesture feedback	One screenshot displays a left-swipe notification.
Blend intensity	The slider reads 100% in the supplied screenshots.
Frame-rate indicator	Individual captures show readings between 15 and 19 FPS.

3. Interpretation

The screenshots are consistent with the design objective of applying an effect to the face and hair while leaving the clothing and classroom background largely natural. The different selected styles produce visibly different colour and texture

treatments. The gesture panel explains how the user can navigate and adjust the output, while the notification communicates that a swipe was detected. These observations provide qualitative evidence of the interface and visual output.

The displayed frame-rate readings are instantaneous values. They should not be interpreted as a sustained average or a benchmark across devices. Frame rate can vary with camera resolution, browser load, inference time, canvas operations and hardware acceleration. A formal performance report would measure processing time over a defined interval and state the hardware and browser configuration.

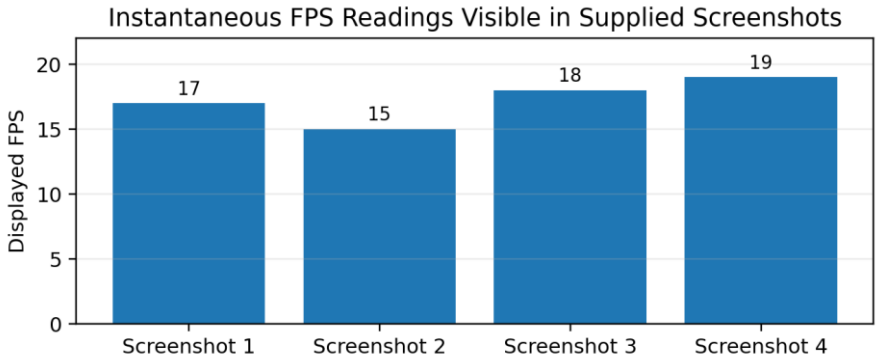


Fig. 6. FPS values transcribed from the four supplied screenshots; these are instantaneous display readings, not a controlled benchmark.

4. Testing and Verification

The supplied project archive includes tests for the gesture engine, vision pipeline, compositor, style rendering and application smoke path. The package test script chains JavaScript tests and a Python rendering test. The presence of these test files is evidence that testing has been considered in the repository; this report does not claim that the complete suite was freshly executed during preparation.

A gesture evaluation should use labelled landmark sequences for each command and for non-gesture movement. It should include slow movement, fast movement, short movement, diagonal movement, stationary jitter, partial occlusion and return strokes. Precision, recall and false triggers per minute can then be calculated. For pinch capture, the false-trigger rate is particularly important because an accidental capture interrupts the user.

Segmentation can be evaluated by comparing the predicted head mask with manually annotated reference masks. Intersection over Union and Dice similarity are suitable overlap measures. A compositor test can use a fixed image and synthetic mask, then verify that pixels outside the mask remain unchanged. A second test can place a hand polygon in front of the head and check that the original hand pixels are preserved.

5. Limitations

The segmentation boundary may change with lighting, camera exposure, head pose, hair texture, hats, glasses and background similarity. Hand tracking may be interrupted by motion blur, partial visibility or an unusual hand orientation. Browser-side model assets require internet access on first load. Performance depends on the device and camera configuration. The optional FLUX mode requires a compatible GPU or hosted API credentials and is not demonstrated as a successful live generation in the supplied screenshots.

VIII. Conclusion And Future Scope

This paper presented Face Gesture Stylizer, a webcam-based application that combines hand landmark tracking, semantic segmentation, artistic image processing and masked compositing. Horizontal swipes navigate the style collection, vertical swipes adjust blend intensity, and a sustained thumb–index pinch requests a photograph capture. The head mask is designed

to confine the visual transformation to hair and facial skin, while the hand cut-out preserves a hand passing in front of the face.

The supplied screenshots demonstrate the live interface and three selected styles: Pop Art Comic, Anime & Manga and Urban Graffiti. They also show the gesture guide, a 100% intensity setting and instantaneous frame-rate readings between 15 and 19 FPS. These observations support the visible operation of the interface but do not establish gesture accuracy, segmentation quality or sustained processing speed. Those properties require controlled evaluation.

Future work should evaluate the system under varied lighting, head poses, backgrounds and camera resolutions; measure gesture precision, recall and false-trigger rate; compare segmentation masks with annotated references; and profile average and percentile frame time on different devices. Further engineering improvements may include mask refinement, more robust hand occlusion handling, configurable gesture thresholds, local model caching and clearer privacy controls for saved captures.

Acknowledgment

I express my sincere gratitude to Dr. Puja Gupta, Department of Information Technology, Shri G. S. Institute of Technology and Science, Indore, for her guidance and academic support during this project. I also acknowledge the developers and maintainers of MediaPipe, OpenCV, FastAPI and the other software components used in the application

References

1. C. Lugaresi et al., “MediaPipe: A Framework for Building Perception Pipelines,” arXiv:1906.08172, 2019.
2. Google, “MediaPipe Hand Landmarker,” MediaPipe Tasks documentation. Available: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker.
3. Google, “MediaPipe Image Segmenter,” MediaPipe Tasks documentation. Available: https://developers.google.com/mediapipe/solutions/vision/image_segmenter.
4. G. Bradski, “The OpenCV Library,” Dr. Dobb’s Journal of Software Tools, 2000.
5. S. Ramírez, “FastAPI,” official documentation. Available: https://fastapi.tiangolo.com/.
6. Black Forest Labs, “FLUX.2 [klein] Model and API Documentation.” Available: https://docs.bfl.ai/.
7. OpenCV, “OpenCV Documentation.” Available: https://docs.opencv.org/.
8. Face Gesture Stylizer project source code and README, supplied project archive, 2026.
9. Face Gesture Stylizer application screenshots, supplied by the project author, November 2, 2026.
   Gupta, P., Sharma, V. and Varma, S., 2021. People detection and counting using YOLOv3 and SSD models. Materials Today: Proceedings.
10. Gupta, P., Shukla, M., Arya, N., Singh, U. and Mishra, K., 2022. Let the Blind See: An AIIoT-Based Device for Real-Time Object Recognition with the Voice Conversion. In Machine Learning for Critical Internet of Medical Things (pp. 177-198). Springer, Cham.
11. Kushwaha, U., Gupta, P., Airen, S. and Kuliha, M., 2022, December. Analysis of CNN Model with Traditional Approach and Cloud AI based Approach. In 2022 International Conference on Automation, Computing and Renewable Systems (ICACRS) (pp. 835-842). IEEE
12. Gupta, P., Sharma, V. and Varma, S., 2022. A novel algorithm for mask detection and recognizing actions of human. Expert Systems with Applications, 198, p.116823.
13. Gupta, P., Sharma, V. and Varma, S., 2022, September. An Algorithm for Counting People using Dense Nets and Feature Fusion. In 2022 4th International Conference on Inventive Research in Computing Applications (ICIRCA) (pp. 1248-1253). IEEE.
14. Gupta, P. and Kulkarni, N., 2013. An introduction of soft computing approach over hard computing. International Journal of Latest Trends in Engineering and Technology (IJLTET), 3(1), pp.254-258
15. Gupta, P., Kulkarni, A. and Sarda, A., 2013. An embedded health care supervisory systems. International Journal of Latest Trends in Engineering and Technology (IJLTET), 3, pp.379-386

16. Jain Monarch,Singar Chandra Prakash , Gupta P, Toxic Speech Classification Using Deep Learning and Machine Learning: A Survey. Journal of Propulsion Technology article 10400
17. Agrawal Neha, Gupta Puja, Barwe Shaivi, Singar Chandra Parakash, Agrawal Deepesh, ESP32-S3 AI Voice Assistant: A Cloud-Augmented Embedded System for Real-Time Voice Interaction. International Journal of Leading Research Publication 2203.
18. Gupta, P, N. Rathore Comparison of cloud and edge computing based face recognition for security enhancement 2025 Journal of Discrete Mathematical Sciences and Cryptography