

Dataset-Free Real-Time Face Recognition: A Browser-Based Implementation Using Deep Facial Embeddings

Metu Okezikam I¹, Dominic Evans O^{2*}, Ihejirika Chijindu Paschal² and Okeke Ogochukwu C¹,

¹ Dept. of Computer Science, Chukwuemeka Odumegwu Ojukwu University, Uli, Anambra State.

²Dept. of Computer Science, Michael Okpara University of Agriculture, Umudike, Abia State

ABSTRACT

Real-time face recognition systems typically depend on large, pre-assembled facial image datasets and dedicated server infrastructure, resources that are often unavailable or impractical for lightweight, on-the-spot deployments. This dependency compounds a longer-standing weakness in identity verification generally: conventional methods such as passwords, PINs, and ID cards remain vulnerable to theft, loss, and misuse, while classical face recognition techniques such as Eigenfaces and Fisherfaces degrade sharply under real-world variation in lighting, pose, and occlusion. This limitation is significant because biometric authentication supports security, surveillance, and access-control systems across sectors such as banking, healthcare, and law enforcement, where both accuracy and real-time responsiveness are essential. This study presents a browser-based, real-time face detection and recognition prototype built entirely with HTML, CSS, and JavaScript, using the face-api.js library (v0.22.2, built on TensorFlow.js) for face detection, landmark localization, and descriptor extraction. Faces are enrolled directly through webcam capture rather than from a public dataset, with each enrollment click generating a 128-dimensional descriptor tied to a user-supplied name; because the number of samples per identity is entirely user-driven, the resulting enrollment set is inherently unbalanced across identities. Recognition is performed by computing the Euclidean distance between a live descriptor and all stored descriptors, applying a threshold (0.35) to accept or reject a match, with enrolled profiles persisted in browser local storage. The system was implemented and functionally tested rather than evaluated against formal classification metrics: no accuracy, precision, recall, or F1-score was computed, and no baseline detector was used for comparison. Real-time performance was assessed by frame-processing rate, with the recognition loop executing every 150 ms, yielding an effective throughput of approximately 6-7 frames per second. Testing confirmed that the prototype reliably performs camera access, live face detection, enrollment, recognition with bounding-box annotation, and persistent profile storage, demonstrating a fully working system rather than a purely conceptual design. The findings show that a lightweight, dependency-free, browser-native approach can deliver functional real-time face enrollment and recognition without server-side infrastructure, though the absence of standardized evaluation metrics and baseline comparison limits claims about its accuracy relative to existing systems..

KEYWORDS;- Face Detection, Face Recognition, Computer Vision, Deep Learning, FaceNet, OpenCV

Date of Submission: 08-10-2026

Date of acceptance: 19-10-2026

I. INTRODUCTION

Identity verification sits at the centre of how security, access control, and public safety are managed today, and the methods relied on for decades, passwords, PINs, ID cards, and other token-based credentials, carry a well-known weakness: they can be forgotten, shared, stolen, or forged. Facial recognition has emerged as an alternative that ties identity to something a person inherently carries rather than something they hold or remember, and improvements in camera hardware and computing power have made it increasingly practical to deploy directly on ordinary consumer devices rather than only in specialized, high-end installations. What once required dedicated servers and purpose-built imaging equipment can now run on a standard laptop or desktop fitted with nothing more than a webcam.

This improvement is pivotal because it lowers the barrier between the technology and the settings where it is actually needed. Applications ranging from attendance tracking and workplace access control to informal identity checks in small organizations increasingly call for a system that works immediately, on hardware already at hand, without the overhead of installing specialized software, provisioning a database server, or assembling a curated dataset before the system becomes useful. Modern browsers, equipped with camera access APIs and capable of running deep learning models directly in JavaScript, present an underused

opportunity in this regard: a face detection and recognition system can, in principle, be built and run entirely within a web page, with no backend server and no separate installation step, while still drawing on the same class of neural network models that power more conventional, server-based systems.

Despite these advances, a number of practical challenges continue to limit how well face detection and recognition systems perform outside controlled conditions. Variations in lighting, pose, distance from the camera, and facial expression remain a persistent source of false positives and false negatives, and systems that are tuned for one environment frequently perform poorly when moved to another. A further, less discussed limitation concerns how such systems acquire the identities they are meant to recognize in the first place. Many existing implementations depend on a pre-assembled dataset of facial images, collected and curated in advance, which must be processed offline before the system becomes usable, a workflow that is difficult to justify for smaller-scale or ad hoc deployments where the people to be recognized are not known ahead of time and a formal data-collection process is impractical. Systems built this way also tend to rely on dedicated backend infrastructure, a server process, a relational database such as MySQL, and often a Python-based computer vision stack involving libraries such as OpenCV, which adds installation and maintenance overhead that is disproportionate to the scale of many everyday use cases.

These limitations point toward a gap that this study sets out to address: the need for a face detection and recognition system that operates in real time, requires no pre-existing dataset, and can be enrolled and used on the spot, directly through a device's webcam, without server-side infrastructure. Rather than assembling a dataset in advance, such a system would allow a user to register their own identity (and that of others) at the point of use, with the system immediately becoming able to recognize the identities it has just been shown. This approach trades the accuracy benefits of a large, curated training set for immediacy and independence from external infrastructure, a trade-off that is appropriate for lightweight, self-contained deployments even though it is not intended to replace large-scale, dataset-driven recognition systems built for high-security applications.

This study therefore aims to design and implement a browser-based, real-time face detection and recognition system that runs entirely on the client side, using JavaScript together with a browser-native face detection and recognition library, and that stores enrolled identities locally on the device rather than in an external database. In pursuit of this aim, the study sets out to:

1. Develop a real-time face detection module that operates on a live webcam feed within a standard web browser, using a JavaScript-based deep learning library rather than a Python or server-side computer vision stack.
2. Implement a webcam-driven enrollment mechanism that allows a user to register facial identities directly at the point of use, without depending on a pre-existing or public facial image dataset.
3. Implement a real-time face recognition mechanism that compares live facial captures against previously enrolled identities and reports a match or non-match with an associated confidence measure.
4. Evaluate the functional behaviour and real-time responsiveness of the resulting system, including its effective frame-processing rate during live recognition.
5. Assess the practicality and limitations of a fully client-side, database-free architecture (one that stores enrolled data in the browser rather than on a server) as an approach to lightweight face detection and recognition.

By pursuing these objectives, the study aims to demonstrate that a functional, real-time face detection and recognition system can be built and deployed without the infrastructural overhead typically associated with such systems, while remaining transparent about the trade-offs, in particular, the absence of a large or balanced training set and the reliance on informal, functional testing rather than dataset-based accuracy evaluation, that this lighter-weight approach necessarily involves.

II. LITERATURE REVIEW

Research on automated face detection and recognition spans two broadly distinct traditions: classical, handcrafted-feature methods optimized for speed, and deep learning approaches optimized for accuracy. The following review is organized around four themes: classical detection methods, deep learning approaches, hybrid & edge-optimized systems, and the ethical issues that cut across all three. The aim is to trace how the field has moved from fast-but-brittle classical detectors toward accurate-but-heavy deep models, and the various attempts to reconcile the two.

2.1 Classical Methods

The Viola-Jones framework remains the reference point for classical, real-time face detection. It combines Haar-like rectangular features, computed efficiently through integral images, with AdaBoost to select the most informative features, arranged into a cascade of classifiers that rejects non-face regions early to save computation. [1] introduced this cascade approach and demonstrated a detector fast enough for the CPUs of its

time, reliably finding frontal faces under controlled conditions, though with limited robustness to pose, scale, and occlusion, a trade-off that later work has confirmed repeatedly.

[2] tested this trade-off directly, comparing Haar cascades against a baseline CNN detector on subsets of FDDB and WIDER FACE. Haar cascades ran at higher frame rates on CPU, but the CNN detector was substantially more precise and more robust to pose and occlusion. [3] pursued a related goal from the landmark-detection side, proposing an ensemble-of-regression-trees approach fast enough for real-time alignment on ordinary CPUs; it remains a common choice where speed matters more than handling difficult cases, webcam applications among them.

Taken together, this line of work establishes the baseline this project measures itself against: classical detectors are cheap and fast but degrade outside controlled conditions, and closing that gap has historically meant reaching for a deep model, alongside the computational cost that comes with it.

2.2 Deep Learning Approaches

Convolutional neural networks changed what was achievable in both detection and recognition. Architectures such as ResNet, which uses residual connections to make very deep networks trainable, and YOLO, which detects objects in a single pass rather than through a separate proposal stage, became the backbone for a generation of face detection systems built for accuracy rather than raw speed.

On the recognition side, the shift toward embedding-based approaches is generally traced to [4], whose FaceNet system used a triplet loss to learn a compact embedding directly optimized for verification, recognition, and clustering, reporting around 99.63% accuracy on LFW. [5] reached comparable territory earlier with DeepFace, combining 3D face alignment with a deep CNN trained on a large proprietary dataset, and [6] contributed VGG-Face as a widely reused off-the-shelf feature extractor. A subsequent wave of work refined the loss functions used to train these embeddings. SphereFace [7] and CosFace [8] introduced angular and cosine margins to push same-identity features closer together and different identities further apart, and ArcFace [9] extended this with an additive angular margin that has since become close to a default choice for high-performance recognition systems. Center loss [10] and L2-constrained softmax [11] pursued the same goal, tighter, more separable feature clusters, through different regularization strategies, while [12] treated triplet sampling itself as a source of instability worth modelling explicitly.

On the detection side, MTCNN [13] popularized joint detection and landmark localization through a three-stage cascade, RetinaFace [14] pushed single-stage detectors toward pixel-level accuracy with dense landmark regression, and [15] showed that aggregating features across scales substantially improves detection of small or partially occluded faces. InsightFace [16] later packaged several of these ideas into a reproducible open-source toolkit that has become a common baseline in its own right.

The pattern across this entire body of work is consistent: accuracy climbs with model depth and training-set size, and so does the computational and data budget required to reach it. Nearly every system discussed above assumes a large, curated dataset and GPU-level inference, assumptions this project's browser-based, dataset-free approach deliberately avoids testing against, in favour of the very different constraint of running with no training data and no dedicated hardware at all.

2.3 Hybrid and Edge-Optimized Systems

A separate strand of research has tried to keep deep learning's accuracy while shedding its computational weight. [17] proposed a two-stage pipeline in which Haar cascades generate fast candidate regions that a CNN then refines, reducing the false positives typical of Haar alone and improving small-face detection, while keeping latency lower than running a CNN over the full image. Subsequently, [18] took a different route to the same goal, designing a compact CNN using depthwise separable convolutions and knowledge distillation from a larger teacher model, reaching near state-of-the-art detection accuracy for its size on mobile hardware. [19] applied pruning, quantization, and backbone reduction to YOLOv4 specifically to fit face detection onto edge devices, and [20] pursued the same efficiency goal on the recognition side with MobileFaceNet, a compact architecture built for mobile-scale accuracy-versus-latency trade-offs.

A further cluster of work extends robustness through additional sensing or processing rather than architecture alone: [21] used thermal imaging to detect faces where visible-light cameras fail in darkness; [22] applied conditional GANs to reconstruct occluded facial regions before recognition; [23] explored face-specific super-resolution to recover recognizable detail from low-resolution surveillance footage; [24] used synthetic, rendered faces to augment training data for underrepresented poses; and [25] combined motion-based tracking with recognition embeddings to maintain consistent identity across frames and cameras. [26] added a security dimension to this cluster, combining depth estimation and remote photoplethysmography signals to distinguish live faces from presentation attacks.

What unites this group of studies is a willingness to trade some accuracy or add engineering complexity in exchange for running on constrained hardware. Even so, nearly all of them still assume a prepared dataset

exists before the system is deployed, and most still assume a dedicated processing environment, whether a mobile GPU, an edge accelerator, or a server. Very little of this literature considers a system with no dataset at all, enrolled entirely at the point of use, which is the specific gap this project occupies.

2.4 Ethical Issues: Bias, Privacy, and Fairness

A parallel body of work has examined the social consequences of deploying facial recognition at scale, rather than its technical performance alone. [27] remains the most cited entry point: their audit of commercial gender-classification systems found error rates far higher for darker-skinned women than for lighter-skinned men, a finding that reframed accuracy benchmarks as incomplete without a breakdown by demographic subgroup. [28] extended this concern to face datasets themselves, showing that the demographic imbalance present in popular training sets carries through directly into uneven recognition accuracy across subgroups, and recommended both more deliberate dataset curation and fairness-aware training procedures.

[29] situated these technical findings within a broader legal and social frame, surveying consent, surveillance, and regulatory issues alongside the recurring observation that many deployed systems offer little transparency about how they were trained or how well they perform across different groups. [30] approached the privacy side of this problem from an engineering angle, using federated learning to keep raw facial data on individual devices and share only aggregated model updates, a design that reduces data-movement risk, though communication overhead and inconsistent data across devices remain open problems, and the strength of the privacy guarantee still depends on the specific protocol used.

This cluster of work is largely diagnostic: it documents where bias and privacy risk enter the pipeline more thoroughly than it prescribes concrete, implementation-level fixes. That gap between naming the problem and building a system that avoids it is picked up again below.

Table 2.1 summarizes the trade-offs discussed across the three technical themes above, alongside where the system built for this study sits in relation to them.

Approach	Representative Works	Typical Accuracy	Speed / Footprint	Infrastructure Needed
Classical (Haar/Viola-Jones)	Viola & Jones (2001); [2]	Moderate; drops sharply with pose/occlusion	Very fast on CPU	None beyond a CPU; no training data at runtime
Deep CNN / embedding-based	[4]; Deng et al. (2019); [5]	High (often >99% on LFW)	Slower; GPU generally assumed	Large labelled training set; often a GPU server
Hybrid / edge-optimized	[17]; [18]; [19]	Moderate-to-high, close to full CNN	Tuned for mobile/edge CPUs	Smaller trained model; still needs a prepared dataset
This project (browser-native, face-api.js)	—	Not formally measured (see Chapter Three)	~6–7 FPS in-browser	None; runs client-side, no dataset, no server

Table 1: Comparison of detection/recognition approaches by accuracy, speed, and infrastructure dependency

2.5 Summary of Literature Review and Knowledge Gap

The literature reviewed above splits fairly cleanly along one axis: systems optimized for accuracy and systems optimized for speed, with comparatively few attempts to reconcile the two without still depending on a curated dataset and dedicated hardware. [4] and the embedding-based work that followed it push recognition accuracy close to its maximum, but only by relying on deep models and large training sets that are impractical to run in real time on ordinary hardware. [2], by contrast, confirms that classical detectors remain fast and cheap, but at a real cost to accuracy under pose variation and occlusion.

Hybrid approaches such as [17] narrow this gap, and edge-optimized work such as [18] and [19] shows that deep models can be compressed enough to run on constrained devices. More importantly for this project, these studies are generally evaluated in controlled settings, and none of them removes the underlying assumption that a labelled dataset must be assembled before the system can be used at all. Separately, the ethics-focused literature [29], [28], [27] is thorough in documenting bias, privacy, and transparency problems, but stops short of proposing implementation-level responses that a small, self-contained system could actually adopt.

From this, three gaps stand out:

- Methodological imbalance: few systems combine deep learning's accuracy with the low latency and low resource cost of classical methods, without falling back on a large training dataset to get there.

- **Hardware-agnostic scalability:** most systems in the literature assume access to a GPU, a mobile accelerator, or at minimum a dedicated processing pipeline, rather than running on whatever ordinary consumer device a user already has.
- **Ethical-operational disconnect:** bias, privacy, and fairness are well documented as problems, but the literature offers comparatively little on how a lightweight, practically deployable system should be built with these concerns already designed in, rather than addressed after the fact.

This project addresses these gaps from a different angle than the hybrid and edge-optimization literature reviewed above. Rather than compressing a deep model to fit constrained hardware, it removes the dataset dependency entirely: the system is built as a browser-native application using JavaScript and the face-api.js library, running detection, landmark localization, and descriptor extraction directly client-side with no server and no external database. Identities are enrolled on the spot through the user's own webcam rather than drawn from a pre-assembled dataset, which sidesteps the data-collection burden common to almost every system discussed in this chapter, at the acknowledged cost of a small, user-defined, and unbalanced enrollment set rather than a large curated one. Because all processing and storage happen on the user's own device, the system also avoids the data-transmission privacy risk that runs through the ethics-focused literature, though it does not attempt to resolve the deeper fairness and bias questions raised by that literature, which remain open questions for future work.

III. METHODOLOGY

3.1 System Overview

The system is a browser-based, real-time face detection and recognition prototype built entirely with HTML, CSS, and JavaScript, running client-side with no backend server and no external database. It is composed of four cooperating modules: a camera input module that captures the live video feed, a detection and description module that locates and encodes faces, a matching module that compares live and stored descriptors, and a browser-based storage module that persists enrolled identities. All four modules execute within a single web page during a single browser session.

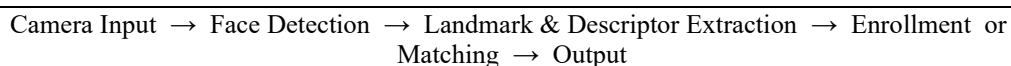


Table 2: System workflow overview

3.2 Enrollment and Data Acquisition

Unlike systems that rely on a curated, pre-existing face dataset, this project does not use any public dataset such as LFW. Enrollment is performed exclusively through the webcam: a user types a name into the interface and clicks "Enroll," and the system captures one facial descriptor from the live video feed at that moment. Because enrollment is entirely user-driven, the number of descriptors associated with a given identity is not fixed or predetermined by the system, it depends solely on how many times the user clicks Enroll for that name. One enrollment click produces one descriptor for the entered name, so identity and sample count are directly coupled to user behaviour rather than to any sampling strategy.

The JavaScript implementation places no cap on the number of enrollments per identity, so the resulting sample set is unbalanced by design: one identity may hold three enrolled samples while another holds only one. The camera feed itself is requested at a fixed resolution of 640×480 pixels using the front-facing camera (getUserMedia with facingMode: "user").

3.3 Detection and Recognition Pipeline

Face detection and description were implemented using face-api.js (version 0.22.2), a JavaScript library built on top of TensorFlow.js, with model weights loaded at runtime from face-api.js's publicly hosted CDN rather than bundled locally, meaning an internet connection is required the first time the page loads. No Python code, OpenCV, or dlib was used at any stage of this implementation.

- **Face Detection:** The TinyFaceDetector locates a face within each video frame. It is configured differently for the two stages of use: during enrollment it runs with a smaller input size (160px) for a fast single-frame capture, while during recognition it runs with a larger input size (224px) to detect all faces present in the frame; both stages use a detection confidence threshold of 0.5.
- **Landmark Alignment:** A 68-point facial landmark model (face-api.js's faceLandmark68Net) aligns each detected face prior to descriptor extraction.
- **Descriptor Extraction:** The library's face recognition network (faceRecognitionNet) produces a 128-dimensional numerical descriptor for each detected face, 128 being the standard output length documented for this model, though the implementation itself does not hard-code this figure and instead reads the descriptor's length dynamically at runtime. This descriptor plays the same conceptual role as a

FaceNet-style embedding, though it is worth noting explicitly that the recognition network is the one bundled with face-api.js rather than a separately trained or fine-tuned FaceNet model. During enrollment, exactly one face is captured per click (detectSingleFace); during recognition, every face present in the frame is detected and compared (detectAllFaces)..

3.4 Matching Procedure

Matching is performed by computing the Euclidean distance between a live descriptor and every descriptor stored for previously enrolled identities:

$$\text{distance}(d_1, d_2) = \sqrt{\sum_i (d_{1i} - d_{2i})^2}$$

where d_1 is the descriptor extracted from the current video frame and d_2 is a stored enrollment descriptor, each a 128-dimensional vector. For each detected face, the system searches across all stored descriptors for every enrolled identity and keeps the smallest distance found. If that smallest distance is at or below a fixed threshold of 0.35, the corresponding name is assigned to the detected face; otherwise the face is labelled "Unknown." This threshold was set deliberately tighter than the 0.45–0.6 range typically cited for face-api.js descriptor matching, favouring fewer false matches at the cost of being more sensitive to variation between enrollment and recognition conditions. No separate classifier such as KNN or SVM is trained, matching is a direct nearest-neighbour distance comparison performed anew on every frame.

3.5 Storage and Persistence

Enrolled profiles, each consisting of a name and its associated descriptors, are persisted using the browser's localStorage API under a single fixed key (face_profiles_v1), whose value is a JSON-serialized object mapping each name to its array of descriptors, rather than a relational database such as MySQL or SQLite. Because face-api.js descriptors are produced as typed Float32Array objects, which cannot be serialized directly, each descriptor is converted to a plain array before being saved and converted back to a Float32Array when reloaded. This means stored data is scoped to the browser and device on which enrollment took place, and no server-side database was implemented in this version of the system.

3.6 Testing Procedure

Testing of the system was carried out through direct, hands-on functional interaction, enrolling sample identities, running recognition against them, and observing the on-screen output, rather than through evaluation on a held-out labelled test dataset. No formal accuracy, precision, recall, or F1-score computation was carried out, and this is treated as a limitation and direction for future work rather than a claimed result. Recognition runs continuously via a polling loop set at a fixed 150-millisecond interval, which the implementation itself notes corresponds to an effective throughput of approximately 6 to 7 frames per second; this is the only performance figure that can currently be stated with confidence, since no baseline detector was used for comparison.

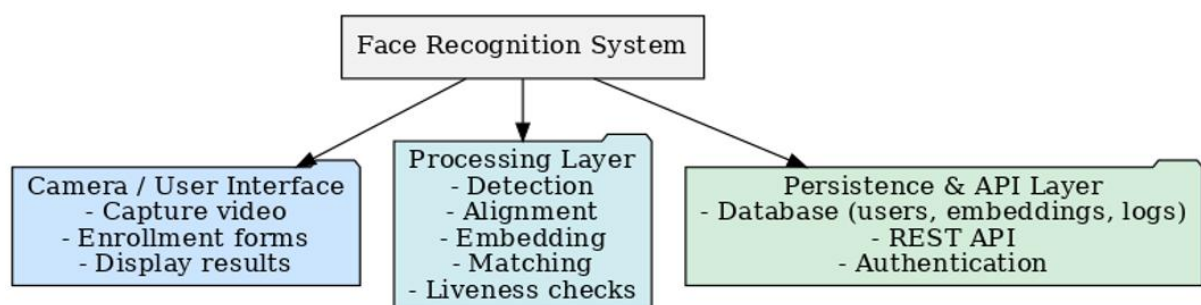


Fig 1: High Level Model Diagram

IV. SYSTEM DESIGN AND IMPLEMENTATION

4.1 Objectives of the Design

The objective of this design is to develop a browser-based Real-Time Face Detection and Recognition prototype that can detect a face from a live webcam feed, allow a user to enroll new identities on the fly, and recognize enrolled identities in real time, entirely on the client side, without a Python backend, external computer vision library, or relational database. The design prioritizes simplicity of deployment and transparency of behaviour over scalability or multi-user administration, both of which are identified as directions for future work rather than features of the current prototype.

4.2 Control Centre/Main Menu

The control centre is a single-page interface arranged in two panels. The left panel holds the live video feed with a canvas overlay for drawing detection results, a name field for enrollment, and three controls: "Enroll (capture)," "Start Recognition," and "Stop," alongside a status line that reports what the system is currently doing (for example, loading models, capturing a face, or running recognition). The right panel holds a log area that opens with built-in usage tips, a live list of saved profiles showing each enrolled name together with its number of stored samples, a "Clear saved profiles" control that removes all enrolled identities after an on-screen confirmation prompt, and a short troubleshooting panel covering camera permissions, HTTPS/localhost requirements, and model-loading issues.

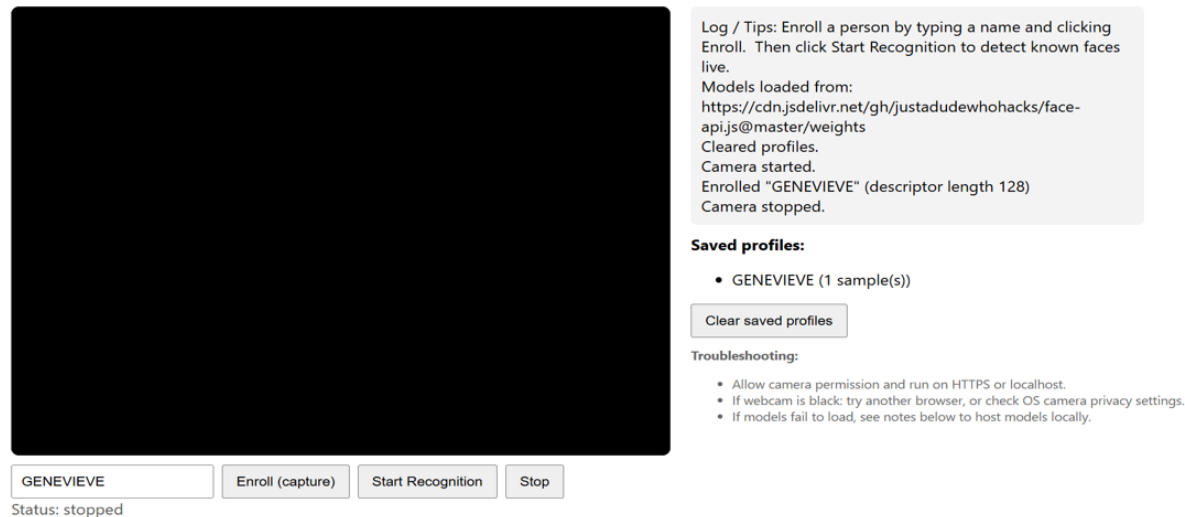


Fig 2: Program Interface Design

4.3 The Submenus/Subsystems

- **Input Module:** acquires the live video stream from the webcam at 640×480 resolution using the front-facing camera, via the browser's getUserMedia API, and can start or stop that stream on demand.
- **Detection Module:** uses face-api.js's TinyFaceDetector to locate faces in the current frame, run with different settings depending on context, a smaller, faster configuration during enrollment and a larger, more thorough configuration during live recognition.
- **Landmark/Feature Extraction Module:** applies a 68-point landmark model and a face recognition network to produce a numerical descriptor for each detected face.
- **Matching/Recognition Module:** compares a live descriptor against every stored descriptor using Euclidean distance and a fixed threshold to decide between a named match and "Unknown."
- **Storage Module:** persists and retrieves enrolled names and descriptors using the browser's localStorage, converting between typed arrays and plain arrays as needed for JSON serialization.
- **Output Module:** renders bounding boxes and identity labels on the canvas overlay, updates the saved-profiles list, and writes status and event messages to the log panel.

4.4 System Specifications

The system is a real-time face detection and recognition prototype that runs entirely in the browser. It captures live camera frames, detects faces, extracts descriptors using face-api.js, and matches those descriptors against locally stored profiles enrolled earlier in the same browser. It supports enrollment and recognition, and includes a manual "clear all profiles" control, but does not include an administrator panel, authentication layer, or persistent audit log, the log panel is held only in page memory and resets whenever the page is reloaded.

4.4.1 Storage Mechanism

The system uses the browser's localStorage API for data persistence rather than a relational database engine such as MySQL. Enrolled profiles are serialized to a single JSON object and saved under one fixed storage key, and are read back and deserialized whenever the page loads.

4.4.2 Storage Design and Structure

Data is stored under a single key, face_profiles_v1, whose value is a JSON object mapping each enrolled name to an array of descriptors. Because face-api.js produces each descriptor as a typed Float32Array, which cannot be serialized directly, every descriptor is converted to a plain array before being written to storage

and converted back to a Float32Array when the page reloads. There are no separate relational tables for users, face data, event logs, or input sources, all enrolled data lives within this one structure, and recognition-log entries are written only to the on-page log panel rather than to persistent storage.

4.4.3 Program Module Specification

- **startCamera / stopCamera:** request or release the webcam stream and, on stop, also clear the recognition loop and canvas.
- **enroll(name):** captures a single face from the current frame, extracts its descriptor, and appends it to the named profile.
- **distance(d1, d2):** computes the Euclidean distance between two descriptors.
- **startRecognition():** begins the repeating detection-and-matching loop against the live video feed.
- **loadProfiles / saveProfiles / renderProfiles:** read and write enrolled profiles to localStorage and keep the on-screen profiles list in sync.
- **log(message):** appends a line to the on-page log panel.
- **init():** loads the required face-api.js models on page start and then restores any previously saved profiles.

4.4.4 Input/Output Format

- **Input:** Live 640×480 video frames captured via the browser's webcam access (MediaStream API).
- **Output:** A canvas overlay showing a bounding box around each detected face with the matched identity (or "Unknown") and the corresponding match distance to three decimal places; enrollment and recognition activity is also echoed to the on-page log, and enrolled identities appear in the saved-profiles list with their sample counts.

4.4.5 Algorithm

- On page load, the required face-api.js models are loaded, after which any previously saved profiles are restored from localStorage.
- On "Enroll": if the camera is not already running, it is started; a single face is then detected in the current frame, its descriptor is extracted, and, if a face was found, the descriptor is added to the entered name's stored samples and saved.
- On "Start Recognition": the camera is started if needed, and every 150 milliseconds every face currently visible in the frame is detected and given a descriptor.
- For each detected face, the Euclidean distance to every stored descriptor is computed and the smallest value is kept; if that value is at or below the fixed threshold, the corresponding name is assigned, otherwise the face is labelled "Unknown," and the result is drawn on the canvas.
- On "Stop," or if the page is closed or navigated away from, the camera stream is released and the recognition loop is cleared.
- On "Clear saved profiles," after an on-screen confirmation, all enrolled names and descriptors are removed from memory and from localStorage.

4.4.6 Data Dictionary

- **name:** String - entered by the user; serves as the identity key under which descriptors are grouped.
- **descriptor:** Array of floating-point numbers - the facial descriptor produced by face-api.js for a single enrollment capture, stored as a plain array converted from a Float32Array for JSON serialization.
- **bestDistance:** Floating-point number - the smallest Euclidean distance found between a live descriptor and the stored descriptors, initialized before comparison and displayed alongside the recognition label.
- **THRESH:** Fixed constant of 0.35 - the maximum distance at which a face is still considered a match.
- **status:** String shown in the status line, describing what the system is currently doing.

4.5 System Flowchart

The system flow is as follows: Load Models → Restore Saved Profiles → then either (Enroll path) Detect Single Face → Extract Descriptor → Save to Storage, or (Recognition path) Detect All Faces Every 150ms → Extract Descriptors → Compare Against Stored Descriptors → Assign Name or "Unknown" → Draw Result.

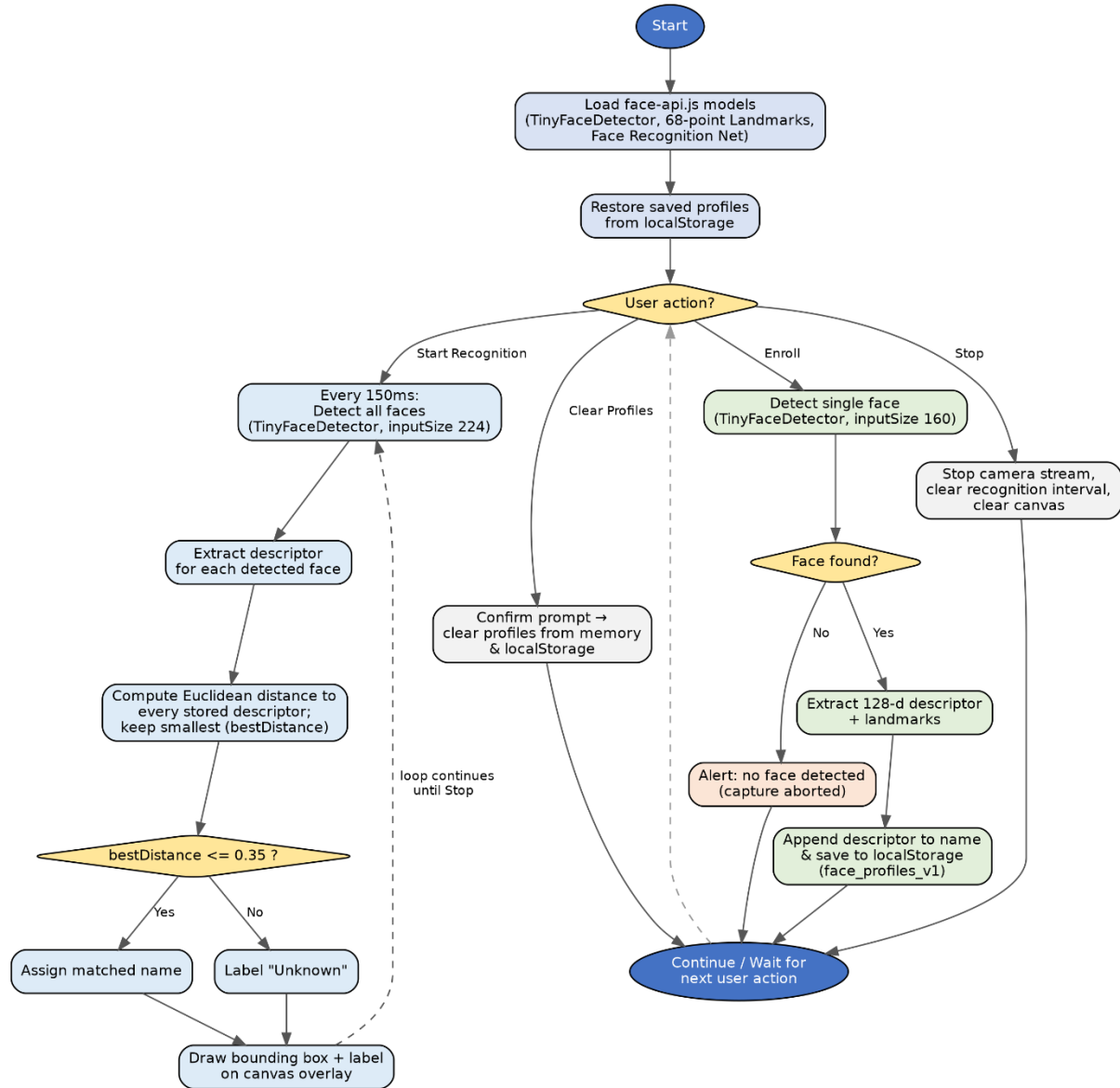


Fig 3: System Flowchart Design

4.6 System Implementation

4.6.1 Hardware Requirements

The implementation does not impose a specific CPU, GPU, or RAM requirement; these are not defined anywhere in the code. In practice, the system runs on whatever computer is used to open it in a browser, provided that computer has a functioning webcam.

4.6.2 Software Requirements

- **Operating System:** Not constrained by the implementation; any OS capable of running a modern web browser is sufficient.
- **Programming Environment:** HTML, CSS, and JavaScript. The face-api.js library itself is loaded from a public CDN, and its model weight files are loaded separately from another public CDN at runtime; the code also notes local hosting of the model files as a fallback option.
- **Connectivity:** An internet connection is required the first time the page loads, to fetch the library and model files; camera access additionally requires the page to be served over HTTPS or from localhost.
- **Libraries:** face-api.js only. No OpenCV, dlib, Python, or server-side machine-learning libraries are used in this implementation.
- **Storage:** Browser localStorage. No MySQL, SQLite, or other database server is used

4.6.3 Program Development

The system is implemented as a single HTML file containing an embedded stylesheet and an embedded script, with no separate server, processing service, database service, or admin panel. The page is laid out in two panels: the video feed, canvas overlay, and enrollment/recognition controls on one side, and the log, saved-profiles list, and troubleshooting notes on the other.

4.6.4 Choice of Program Environment

JavaScript, HTML, and CSS were used for the entire implementation. JavaScript was chosen specifically because face-api.js provides ready-to-use, browser-native face detection, landmark localization, and descriptor extraction without requiring a Python environment, server setup, or native computer vision libraries such as OpenCV or dlib, neither of which appears anywhere in the codebase.

4.6.5 Enrollment (Face Capture)

There is no offline training or dataset-preparation stage in this system. What functions as "training" is, in practice, enrollment: the user types a name, clicks Enroll, and the system captures one descriptor from the live camera feed for that name using a fast, small-input-size detection pass. Clicking Enroll again for the same name captures an additional descriptor and adds it to that identity's stored samples. Because this process is entirely driven by user interaction rather than a fixed sampling procedure, the number of descriptors per identity is unpredictable in advance and typically unbalanced across identities.

4.6.6 Documentation

User-facing documentation is built directly into the interface rather than kept in a separate manual: the log panel opens with basic usage tips, and a troubleshooting section covers camera permissions, the HTTPS/localhost requirement, and steps to take if the webcam feed fails or the models do not load. Technical documentation for this report covers the single-file architecture, the localStorage data structure, and the detection/matching logic described above.

4.7 Testing

Testing of the prototype was functional rather than metric-based. The following behaviours were manually verified and confirmed working: webcam access under the HTTPS/localhost requirement, real-time face detection with bounding-box rendering, enrollment of new identities via webcam capture, recognition of enrolled identities with the correct name and distance value displayed, the "Clear saved profiles" control together with its confirmation prompt, persistence of enrolled profiles in localStorage across page reloads, and automatic release of the camera when the page is closed or navigated away from. This confirms the system as a fully implemented, working prototype rather than a conceptual design.

No formal unit, integration, or performance test suite was implemented. Frame rate was not benchmarked with dedicated tooling but follows directly from the fixed 150-millisecond recognition polling interval used in the code, corresponding to an effective throughput of approximately 6 to 7 frames per second. No accuracy, precision, recall, or F1-score was computed against a labelled test set, and no comparison against a baseline detector was carried out in this version of the work.

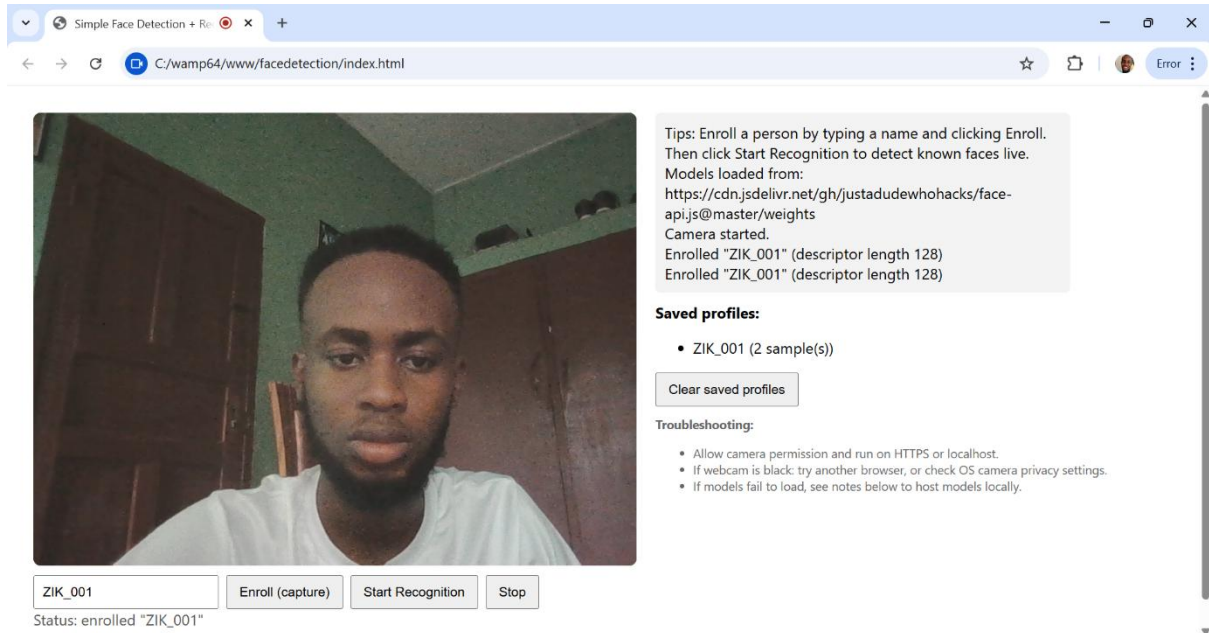


Fig 4: Face Enrollment/Capturing

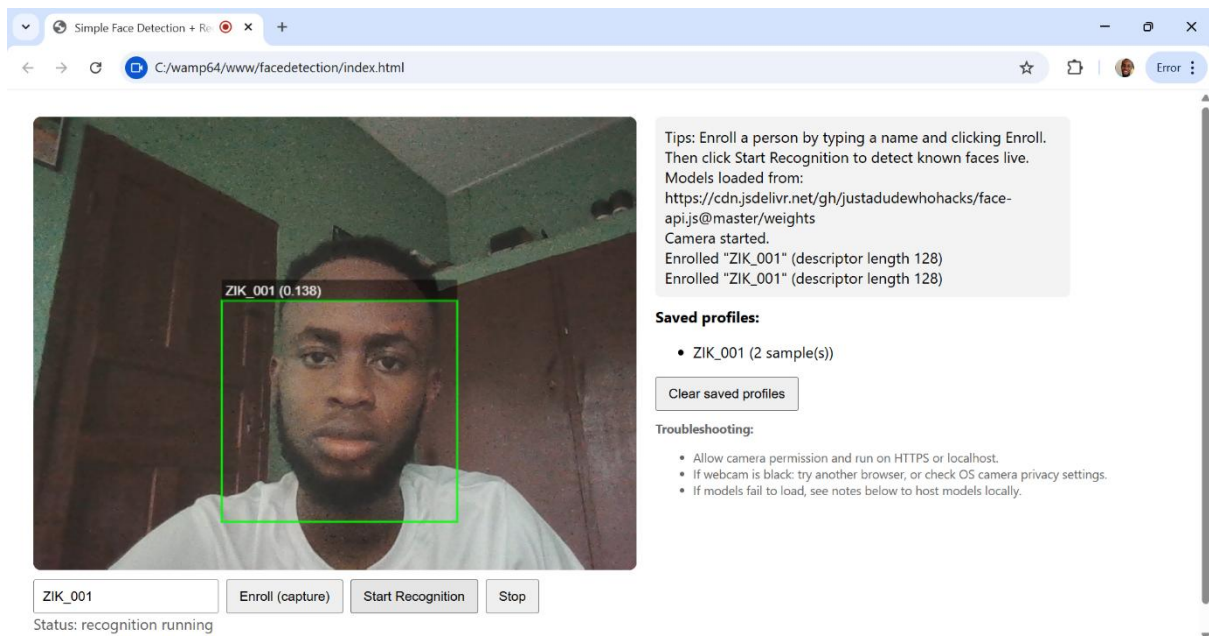


Fig 5: Face Enrolled/Captured Successfully

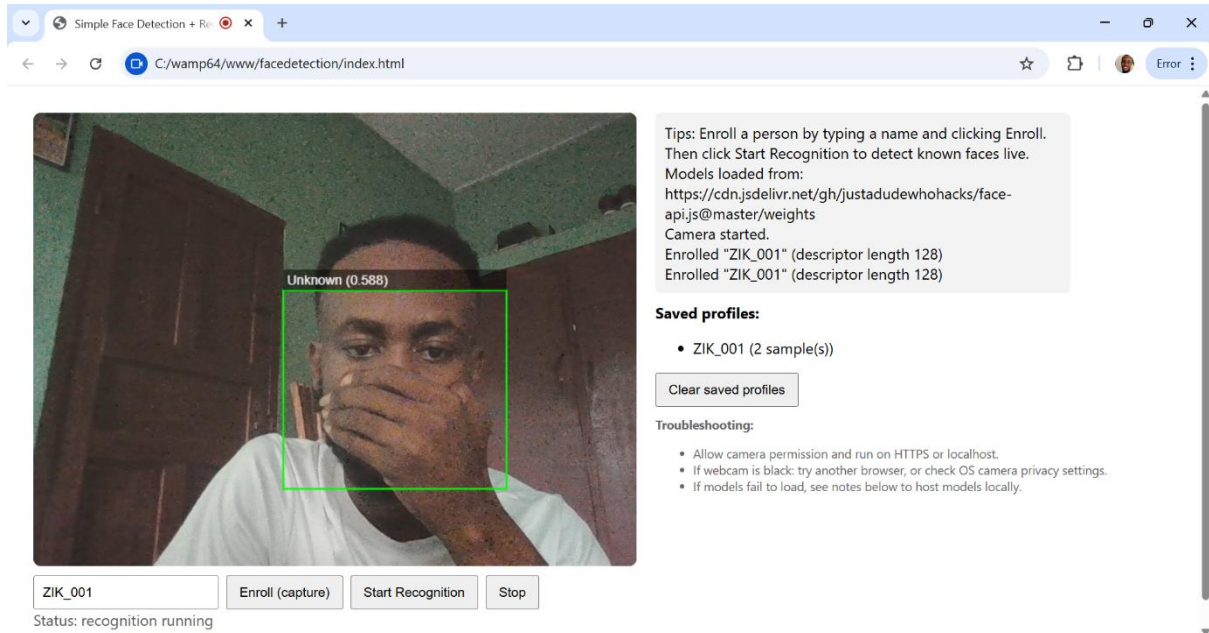


Fig 6: Face Enrollment/Capturing Failure due to obstruction

V. DISCUSSION

The implementation described above demonstrates that a working, real-time face detection and recognition system can be built without any of the infrastructure the literature reviewed in Chapter Two generally assumes: no pre-assembled dataset, no server, no relational database, and no Python-based computer vision stack. Where hybrid and edge-optimized approaches such as [17] and [18] reduce computational cost by compressing or restructuring a model that still needs to be trained on a prepared dataset, this system removes the dataset requirement altogether by enrolling identities directly from the webcam at the point of use. That is a genuinely different position in the design space discussed in Chapter Two, rather than a smaller version of the same approach, and it directly addresses the hardware-agnostic scalability gap identified in Section 2.5: the system runs on whatever consumer device is available, with no GPU or dedicated processing pipeline assumed anywhere in the code.

The matching threshold is worth discussing on its own terms. The implementation fixes the acceptance threshold at 0.35, tighter than the 0.45 to 0.6 range typically cited for face-api.js descriptor matching, which the code itself notes in a comment. This was a deliberate choice favouring fewer false matches over fewer false rejections, but it has not been validated empirically: because no accuracy, precision, or recall was computed, it is not currently known how often this stricter threshold causes an enrolled person to be labelled "Unknown" under lighting or pose conditions that differ from their enrollment capture. This is flagged here as an assumption built into the system rather than a demonstrated result, and is a natural starting point for future evaluation.

Storing descriptors exclusively in browser localStorage, rather than transmitting them to a server, gives the system a genuine privacy property in line with the concerns raised by [29] in the ethics literature reviewed in Chapter Two: facial data never leaves the device on which it was captured. This is a narrower guarantee than the federated-learning approach of [30], which still allows a model to be improved across users while keeping raw data local; this system does not attempt any cross-device learning or synchronization at all, so the privacy benefit comes at the cost of each enrollment being usable only on the browser and device where it was created. The system also does not attempt to address the fairness and demographic-bias concerns raised by [27] and [28], since its enrollment set is small, self-selected, and was never evaluated across demographic subgroups; any claim about consistent performance across users would be speculative given the testing carried out so far.

Three limitations follow directly from the design choices described in this chapter and are worth stating plainly rather than glossing over. First, the enrollment set is unbalanced by construction, since the number of samples per identity depends entirely on how many times a user clicks Enroll, with no minimum or maximum enforced. Second, the system has been evaluated only through informal, hands-on functional testing rather than against a labelled test set, so the 6 to 7 frames-per-second figure is the only performance claim that can currently be made with confidence; no accuracy-related figure should be inferred from this work. Third, because storage is scoped to a single browser and device, the system offers no way to recognize an enrolled identity from a different device, and clearing browser data removes all enrolled profiles with no recovery path. None of these are flaws introduced during implementation; they are direct, traceable consequences of the client-side, dataset-

free architecture chosen for this prototype, and each is a reasonable target for future work rather than an oversight in the current one.

VI. CONCLUSION

This study set out to design and implement a browser-based, real-time face detection and recognition system that runs entirely on the client side, requires no pre-existing dataset, and can be enrolled and used on the spot through a device's webcam. Each of the objectives set out in Chapter One was addressed directly. A real-time detection module was implemented using face-api.js's TinyFaceDetector, running inside a standard web browser rather than a Python or server-side computer vision stack. A webcam-driven enrollment mechanism was built that lets a user register an identity at the point of use, with no dependency on a public facial image dataset. A recognition mechanism was implemented that compares live descriptors against enrolled ones using Euclidean distance and reports a match, or "Unknown," together with a distance value that serves as its confidence measure. The system's functional behaviour and real-time responsiveness were evaluated, establishing an effective throughput of approximately 6 to 7 frames per second from its fixed 150-millisecond recognition interval. Finally, the practicality of a fully client-side, database-free architecture was assessed directly by building and testing one: the completed prototype runs with no external server, no relational database, and no Python dependency of any kind.

Taken together, these results show that a functional, real-time face recognition system does not need the infrastructure such systems are usually built with. Detection, enrollment, and recognition all run inside a single web page, with enrolled identities stored only in the browser that created them. This does not replace large-scale, dataset-trained recognition systems built for high-security use, nor does it claim to match their accuracy; what it demonstrates is a different, lighter-weight point in the same design space, one traded deliberately for immediacy and independence from external infrastructure rather than for maximum accuracy. The trade-offs that come with that choice are addressed directly, not glossed over, in the section that follows.

5.1 Limitations of the Study

- Enrollment samples per identity are unbalanced, since the number depends entirely on how many times the user clicks Enroll, with no minimum or maximum set.
- No formal accuracy testing was carried out; the system was verified functionally rather than measured against a labelled test set.
- Enrolled profiles are stored only in the browser used for enrollment, so they don't carry over to a different device or browser.

5.2 Recommendations

The following are recommended as direct next steps building on the limitations identified above:

- Enforce a minimum number of enrollment samples per identity to reduce imbalance.
- Evaluate the system against a labelled test set to establish accuracy, precision, and recall.
- Add a way to export and import enrolled profiles across devices or browsers.

REFERENCES

- [1] P. Viola and M. J. Jones, "Rapid Object Detection Using a Boosted Cascade of Simple Features," *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 511–518, 2001. DOI: 10.1109/CVPR.2001.990517.
- [2] N. A. B. M. Ariffin, U. A. Gimba, and A. Musa, "Face Detection Based on Haar Cascade and Convolution Neural Network (CNN)," *Journal of Advanced Research in Computing and Applications*, vol. 38, no. 1, pp. 1–11, 2025. DOI: 10.37934/arca.38.1.111.
- [3] V. Kazemi and J. Sullivan, "One Millisecond Face Alignment with an Ensemble of Regression Trees," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1867–1874, 2014. DOI: 10.1109/CVPR.2014.241.
- [4] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 815–823, 2015.
- [5] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1701–1708, 2014.
- [6] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep Face Recognition," *Proceedings of the British Machine Vision Conference (BMVC)*, 2015.
- [7] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, "SphereFace: Deep Hypersphere Embedding for Face Recognition," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 212–220, 2017.
- [8] H. Wang, Y. Wang, Z. Zhou, X. Ji, D. Gong, J. Zhou, Z. Li, and W. Liu, "CosFace: Large Margin Cosine Loss for Deep Face Recognition," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5265–5274, 2018.
- [9] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, "ArcFace: Additive Angular Margin Loss for Deep Face Recognition," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4690–4699, 2019.
- [10] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, "A Discriminative Feature Learning Approach for Deep Face Recognition," *European Conference on Computer Vision (ECCV)*, pp. 499–515, 2016.

- [11] R. Ranjan, C. D. Castillo, and R. Chellappa, "L2-Constrained Softmax Loss for Discriminative Face Verification," arXiv:1703.09507, 2017.
- [12] S. Sankaranarayanan, A. Alavi, C. D. Castillo, and R. Chellappa, "Triplet Probabilistic Embedding for Face Verification and Clustering," 2016 IEEE 8th International Conference on Biometrics Theory, Applications and Systems (BTAS), 2016. DOI: 10.1109/BTAS.2016.7791205.
- [13] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks," IEEE Signal Processing Letters, vol. 23, no. 10, pp. 1499–1503, 2016.
- [14] J. Deng, J. Guo, E. Ververas, I. Kotsia, and S. Zafeiriou, "RetinaFace: Single-Shot Multi-Level Face Localisation in the Wild," Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5203–5212, 2020.
- [15] T. Wu, D. Liang, J. Pan, H. Sun, B. Kang, S. Kaneko, and H. Zhou, "Score-Specific Non-Maximum Suppression and Coexistence Prior for Multi-Scale Face Detection," Proceedings of ICASSP, 2019.
- [16] J. Deng, J. Guo, X. An, Z. Zhu, and S. Zafeiriou, "Masked Face Recognition Challenge: The InsightFace Track Report," IEEE/CVF International Conference on Computer Vision Workshops (ICCVW), 2021.
- [17] T. T. Thai and D. T. Nguyen, "Face Detection Method in Surveillance Systems Using Haar Feature and Deep Neural Network," 2019 6th NAFOSTED Conference on Information and Computer Science (NICS), pp. 434–438, 2019. DOI: 10.1109/NICS48868.2019.9023868.
- [18] H. Jin, S. Zhang, X. Zhu, Y. Tang, Z. Lei, and S. Z. Li, "Learning Lightweight Face Detector with Knowledge Distillation," 2019 International Conference on Biometrics (ICB), pp. 1–7, 2019. DOI: 10.1109/ICB45273.2019.8987309.
- [19] X. Wu et al., "Face Detection Method Based on Lightweight Network and Weak Semantic Segmentation Attention Mechanism," 2022.
- [20] S. Chen, Y. Liu, X. Gao, and Z. Han, "MobileFaceNets: Efficient CNNs for Accurate Real-Time Face Verification on Mobile Devices," Chinese Conference on Biometric Recognition (CCBR), pp. 428–438, 2018. DOI: 10.1007/978-3-319-97909-0_46.
- [21] H. Zhang, V. M. Patel, B. Riggan, and S. Hu, "Generative Adversarial Network-Based Synthesis of Visible Faces from Polarimetric Thermal Faces," International Joint Conference on Biometrics (IJCB), 2017.
- [22] F. Zhao, J. Feng, J. Zhao, W. Yang, and S. Yan, "Robust LSTM-Autoencoders for Face De-Occlusion in the Wild," IEEE Transactions on Image Processing, vol. 27, no. 2, pp. 778–790, 2018. DOI: 10.1109/TIP.2017.2771408.
- [23] X. Yu, F. Shiri, B. Ghanem, and F. Porikli, "Can We See More? Joint Frontalization and Hallucination of Unaligned Tiny Faces," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 9, pp. 2148–2164, 2020. DOI: 10.1109/TPAMI.2019.2914039.
- [24] I. Masi, S. Rawls, G. Medioni, and P. Natarajan, "Pose-Aware Face Recognition in the Wild," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4838–4846, 2016.
- [25] X. Zhou, K. Jin, Q. Chen, M. Xu, and Y. Shang, "Multiple Face Tracking and Recognition with Identity-Specific Localized Metric Learning," Pattern Recognition, vol. 75, pp. 41–50, 2018. DOI: 10.1016/j.patcog.2017.09.022.
- [26] S. Zhang, X. Wang, A. Liu, C. Zhao, J. Wan, S. Escalera, H. Shi, Z. Wang, and S. Z. Li, "A Dataset and Benchmark for Large-Scale Multi-Modal Face Anti-Spoofing," Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 919–928, 2019.
- [27] J. Buolamwini and T. Gebru, "Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification," Proceedings of Machine Learning Research, vol. 81, pp. 77–91, 2018.
- [28] A. K. Jain, K. Nandakumar, and A. Ross, "50 Years of Biometric Research: Accomplishments, Challenges, and Opportunities," Pattern Recognition Letters, vol. 79, pp. 80–105, 2016. DOI: 10.1016/j.patrec.2015.12.013.
- [29] X. Wang, Y. C. Wu, M. Zhou, and H. Fu, "Beyond Surveillance: Privacy, Ethics, and Regulations in Face Recognition Technology," Frontiers in Big Data, vol. 7, 1337465, 2024. DOI: 10.3389/fdata.2024.1337465.
- [30] X. An et al., "Federated Learning for Face Recognition with Gradient Correction," Proceedings of the AAAI Conference on Artificial Intelligence, 2022.