

# Smart Edge Project Plan

Sadat Kabir

**Abstract** - Biometric access systems have emerged as a secure alternative to conventional mechanical and token-based locking mechanisms; however, most existing solutions depend on cloud connectivity, exhibit high latency, or lack resilience against spoofing and tampering. This project presents SmartEdge Access, an edge-computing biometric smart lock that performs on-device facial recognition, liveness verification, and secure actuation of a physical locking mechanism. The system integrates embedded computer vision, a lightweight machine-learning inference pipeline, and a deterministic control architecture to ensure reliable authentication without reliance on remote servers. A microcontroller-driven actuation subsystem, augmented by real-time feedback and fail-safe control logic, enforces secure lock transitions, while a companion administrative interface provides user management and audit logging capabilities. Security considerations—including encrypted credential storage, tamper detection, and fallback authentication modes—are incorporated to harden the device against adversarial manipulation. Experimental evaluation demonstrates that the proposed system achieves low-latency unlock operations and maintains authentication accuracy suitable for residential and commercial use cases. The resulting design constitutes a self-contained, privacy-preserving, and scalable access control platform, representing a significant step toward robust, intelligent, and deployable edge-based physical security systems.

*Index Terms* - Internet of Things (IoT), Digital Signal Processing (DSP), Machine Learning (ML), Computer Vision (CV).

## INTRODUCTION

Biometric authentication has increasingly become a preferred alternative to traditional access control mechanisms, driven by advances in embedded machine learning, low-power computing platforms, and ubiquitous personal identification technologies. Despite this progress, existing smart lock solutions remain heavily dependent on cloud-based verification pipelines, external devices such as smartphones or Bluetooth tokens, and weakly integrated mechanical actuation systems. These limitations introduce latency, privacy leaks, and system fragility, particularly in scenarios where network connectivity cannot be guaranteed or where adversaries may exploit unsecured communication channels to bypass authentication checks. Furthermore, many commercially available systems provide minimal protection against biometric spoofing, lack deterministic

lock-state behaviors, and offer insufficient traceability of physical access events.

To address these deficiencies, this paper proposes **SmartEdge Access**, an edge-based biometric smart lock capable of performing all authentication and inference operations locally. The system integrates embedded computer vision, a lightweight facial recognition model optimized for microcontroller-class devices, and a secure actuation subsystem governed by a formally defined control logic. By executing identity verification and liveness assessment directly at the edge, the design ensures that unlocking decisions occur without dependence on external servers, thereby reducing latency and eliminating exposure of biometric information beyond the physical device. In addition, the architecture incorporates encrypted storage for credentials, administrative tooling for user management, configurable fallback pathways, and a logging subsystem that records access events for audit and diagnostic purposes. The contributions of this work are fourfold: the development of a privacy-preserving, self-contained biometric access control device; the unification of machine learning inference, cyber-physical actuation, and embedded system reliability into a deployable architecture; the integration of tamper-aware mechanisms and security hardening measures suitable for real-world environments; and an empirical demonstration of system performance showing low-latency operation and high authentication fidelity. The remainder of this paper is structured as follows. Section II examines related work and existing approaches to smart access control. Section III details the proposed system architecture and its constituent modules. Section IV presents the experimental methodology and evaluation setup. Section V discusses the results and deployment considerations. Section VI concludes the paper and outlines future research directions.

## PROBLEM STATEMENT

The rapid adoption of smart access solutions has not been matched by a corresponding evolution in their security and architectural robustness. Existing commercial offerings rely predominantly on mobile tokens, RF-based credentials, or cloud-mediated verification, creating persistent vulnerabilities that compromise both privacy and reliability. Cloud dependency introduces single points of failure and exposes biometric data to off-device storage and processing, while token-based systems are prone to duplication, interception, and unauthorized transfer between individuals. Moreover, current biometric locks rarely incorporate liveness detection, leaving them susceptible to spoofing attacks using printed photographs or digital replay. Their

mechanical actuation subsystems frequently lack deterministic control behavior, providing no guarantees regarding the lock's response under edge-case conditions, fault states, or asynchronous inputs. These deficiencies underscore a fundamental gap in the availability of **self-contained, tamper-resistant, and verifiably secure biometric locks** capable of independent authentication, real-time physical actuation, and accountable access traceability without reliance on external infrastructure or user-held devices.

## SYSTEMS ARCHITECTURE

At a high level, the system is this cyber-physical access control:

Camera + sensors → DSP/CV/ML at the edge → control logic → motor/lock → logs + dashboard via backend/frontend → tested & secured via CI/CD, security and (optionally) formal methods. Although the system maintains standalone operational capability, large-scale deployments benefit from a distributed mesh communication fabric that enables locks to exchange access events, local policy decisions, and health telemetry without reliance on a central backend. This can be realised through lightweight gossip-based synchronisation or Conflict-Free Replicated Data Types (CRDTs), allowing devices to converge on a shared access history and enforce consistency during periods of network isolation. Such peer-to-peer cooperation transforms isolated locks into a coordinated perimeter security network rather than a collection of independent endpoints.

## HARDWARE STACK

In the first iteration, the hardware is almost embarrassingly simple: an ESP32-CAM board acts as both the compute unit and the camera interface, powering a cheap plastic servo or a small solenoid directly from GPIO control [2] [3]. Power delivery is handled via a generic USB supply or a 5 V regulator, and all wiring lives on a breadboard or perfboard. There is no custom PCB, no isolation circuitry, and no serious EE considerations—just an MCU, a camera, and a motor being told when to move.

By the second iteration, that minimalism disappears. The system now pivots to a Raspberry Pi (or a Pi Zero 2W) as the main compute node, with the camera separated from control electronics. The lock hardware becomes a purpose-built commercial door strike or geared actuator, driven through a proper MOSFET stage with flyback diodes and a buck converter capable of stepping down from 12 volts to logic rails. Limit switches begin to appear, giving the device positional awareness. The ESP32 may remain, but only as an IO co-processor. A small KiCad-designed PCB replaces the rat's nest of jumper wires, anchoring the power distribution, connectors, and driver electronics.

In the third iteration, the hardware stops being hobbyist and starts resembling a product. The Raspberry Pi or Jetson Nano stays on the high-compute side, but now delegates

motor control and sensor supervision to a dedicated MCU such as an STM32 or a more structured ESP32 board. Additional sensors—RFID readers, PIR presence detectors, tamper switches, microphones, and accelerometers—join the system [1]. The PCB grows into a multi-functional IO board with surge protection, ESD suppression, and correct trace widths for driving lock hardware. The enclosure shifts from a plastic box into metal or reinforced composite that resists casual prying.

In the fourth and final iteration, the hardware matures into something that could be manufactured and sold. The logic is consolidated into a custom PCB that integrates compute, IO, secure element, power conversion, connectors, and lock drivers into a coherent electrical design. The tamper switch becomes part of a mesh or intrusion-detecting arrangement, and the case itself participates in security. The electrical architecture observes grounding, isolation, EMI considerations, and surge pathways. At this point, nothing about the hardware looks like a DIY attempt; it resembles a commercial board you would expect inside a professionally engineered edge-security device.

After the fourth iteration's commercial-grade board, the fifth evolution adds computational silicon that no ordinary access control device possesses: a **hardware neural accelerator**. The Raspberry Pi or Jetson is no longer the only inference engine—now a small FPGA board or a custom ASIC implements a face embedding or keyword spotting pipeline in hardware. Frames from the camera feed into DMA channels that deliver pixel streams directly to fabric logic, bypassing CPU bottlenecks. Neural weights sit in block RAM, quantized and compressed, enabling inference at latencies unattainable in software. At this point the PCB is redesigned to host the accelerator directly or exposed via PCIe, MIPI, or parallel memory buses. Instead of “a lock with ML,” the hardware becomes an edge device whose very identity is defined by silicon-accelerated biometrics. This is the point where Cadence tools begin making sense: Spectre or Virtuoso for analog front ends, Innovus for physical layout, and perhaps Allegro for a multi-layer PCB that stitches the lock, sensor front ends, and neural fabric into a coherent board.

The sixth hardware iteration takes a darker turn: the lock becomes physically self-aware. This is where a **secure enclave** enters the design—either a dedicated ATECC-class trust anchor or a tiny TPM-like platform that holds cryptographic secrets unreachable by software compromise. The enclosure gains tamper detection that does not merely sense an opened lid but measures continuity across a conductive mesh, detects drilling, and responds to invasive attacks by erasing keys. Now the board is not only a compute platform but a vault.

The seventh hardware leap introduces **custom sensor ASIC pathways**. If audio becomes significant, the MEMS microphone interface migrates from generic ADC inputs to a tailored DSP front-end on silicon—anti-aliasing filters, automatic gain control, and MFCC preprocessing

implemented in transistors instead of C. If gait recognition is added, the IMU path could similarly gain a custom ASIC front-end for high-frequency motion capture. At this stage, the hardware is no longer composed of modules; it becomes an ecosystem of silicon primitives whose behaviour embodies the system's identity.

Although the electromechanical evolution of the lock hardware is described in detail, real-world deployment requires addressing mechanical reliability considerations that govern safety certifications and regulatory acceptance. Actuator stall current, thermal dissipation margins, and expected lifetime cycles must be characterised to prevent hardware degradation under repeated load. Policies for power-failure scenarios—such as whether the lock fails secure or fails open—must align with legal requirements for egress under fire and emergency conditions. These mechanical constraints elevate the access system from a functional prototype to a compliant physical security mechanism suitable for residential and commercial integration.

### EMBEDDED, OS, AND CONTROL STACK

The first iteration runs as a single blocking loop—one chunk of C/C++ code on the ESP32-CAM, checking for a face and then toggling the lock. There is no notion of state, no concurrency, and no safety. The code simply reacts, delays, and resets. Failure modes consist of jammed motors and hope.

In iteration two, the software architecture becomes aware of behaviour. On a Raspberry Pi, a background service (written in Python or C++) talks to the GPIO subsystem, manages recognition results, and implements a rudimentary finite state machine: detect face, verify, unlock, wait, relock. Timing is no longer ad-hoc; actions are coordinated. The ESP32, if present, is demoted from primary compute to a peripheral supervisor handling IO.

Iteration three is where embedded software earns the word *embedded*. The Pi or Jetson runs the ML and business logic, but the MCU runs FreeRTOS or a well-structured scheduler, ensuring real-time guarantees for the lock mechanism. The state machine is complete and defensive, handling resets, invalid transitions, and mechanical feedback. PID control or current-feedback actuation appears, not to be cute but to eliminate jitter and oscillation when driving a solenoid or servo. Hardware watchdogs, dead-time protection, and jam-detection logic become part of normal operation.

In iteration four, the embedded software stack is no longer a clever script—it is a verified system. The finite-state machine that governs unlocking and relocking is formally specified, reviewed, and checked through tools like TLA+ or equivalent assertions. Firmware images are signed; secure boot prevents arbitrary overwrites; and lock actuation runs in a fault-tolerant manner independent of the Linux subsystem. At this level, failure modes are classified, tested, and mitigated.

Beyond iteration four, the MCU stops being a smart GPIO liaison and becomes a scheduler for multiple biometric

pipelines. Unlock decisions are no longer binary events—they are weighted conclusions derived from several time-dependent streams: facial embeddings stabilized over a sequence, microphone features aggregated over windows, proximity data, and RFID signatures. The firmware moves from a deterministic FSM to a **temporal context engine**, where identity is validated over a horizon of observations. The embedded software now performs confidence scoring, identity decay functions, and fallback arbitration logic instead of simple “face recognized → unlock”. Suddenly, the door begins acting like autonomous systems do: it forms beliefs, discards anomalies, and refuses to unlock until sensor consensus is achieved.

With the secure enclave introduced, the control plane now participates in attestation. Commands issued by the Linux subsystem are not trusted without signatures, and firmware itself refuses to run unless authenticated. State transitions are hashed, logged, and signed. The door now has a memory and a truth model.

When behavioural analytics arrive, the MCU learns to recognize not just *who* approaches, but *how* they approach. Access decisions can be delayed, escalated, or denied based on temporal behaviours—loitering, gaze inconsistencies, abnormal approach velocity. Control logic ceases to be reactionary; it becomes predictive.

While the system architecture incorporates a real-time operating environment, future iterations require explicit determinism guarantees to ensure safety under all operating conditions. Critical control paths—such as facial recognition inference, lock actuation, and watchdog-triggered recovery workflows—must be bounded by strict latency budgets. This necessitates a formal Worst-Case Execution Time (WCET) analysis of the recognition loop, including maximum inference time per frame, deadlines for motor engagement and disengagement, and the permissible reset window before a watchdog intervention occurs. By quantifying temporal bounds rather than assuming best-case performance, the system transitions from a responsive prototype to a cyber-physical controller capable of predictable, verifiable behaviour under timing stress, adversarial load, or degraded resources.

### ML, CV, DSP, FUSION STACK

Iteration one barely deserves the ML acronym. Face recognition is provided by pre-canned ESP32 code or OpenCV's Haar cascades, with identity stored as raw templates. No real embeddings, no real understanding—just pattern comparison.

Iteration two replaces the toy recogniser with embedding-based verification. A lightweight FaceNet or dlib embedding model creates numerical identity vectors, stored locally and compared using cosine similarity. Recognition is resilient to lighting changes, and image preprocessing begins—perhaps histogram equalisation or alignment. DSP still plays no role.

Iteration three transforms ML from a gimmick into an instrument. Detection shifts to RetinaFace, MTCNN, or a trimmed YOLO model, while embeddings move to ArcFace

or InsightFace, optimised and exported to ONNX or TensorFlow Lite for edge inference. Liveness detection emerges—blink recognition, head-pose validation, temporal consistency checks. DSP enters: microphones feed I<sup>2</sup>S audio into an MCU or Pi, MFCC features are extracted, and a tiny keyword spotting model augments face verification with voice cues. Suddenly, recognition is no longer visual—it is multimodal.

Iteration four hardens the above rather than embellishes it. Models are quantised, pruned, accelerated via TensorRT or TFLite delegates, and validated statistically. Liveness evolves from “detect a blink” into multi-cue spoof resistance, possibly with IR texture cues. Voice unlock becomes noise-robust or disappears if deemed insecure. The stack stops growing and begins ossifying into something deployable.

The fifth iteration replaces the singular modality of face recognition with **biometric fusion**. Instead of “your face lets you in,” the system evaluates multiple signals—voiceprints extracted via MFCCs, RFID identity tags, PIN keypad input, and even gait signatures inferred from body motion. Each modality produces a confidence value; a fusion model combines them into a canonical access score. Reliability becomes probabilistic, not boolean. Decisions now encode uncertainty, reminiscent of Kalman filters and sensor fusion stacks in robotics. The system no longer recognizes you; it understands you.

Once the hardware accelerator appears, inference latency collapses. Instead of running FaceNet or ArcFace on a CPU, embeddings stream through FPGA fabric, producing vectors faster than frames can land. Keyword spotting becomes instant—DSP blocks deliver MFCCs into neural nets without touching Linux. When someone speaks the unlock phrase while presenting their face, both pipelines fire at silicon speed and fuse upstream.

With behavioural analytics, CV evolves a third dimension—time. Faces are tracked, not sampled. Gaze is inferred and matched against expected patterns. Suspicious dwell-time is flagged. The lock now discerns *intent*. It can tell the difference between an employee walking normally and someone lingering uncertainly.

This stack ends only when the ML ceases being a classifier and becomes a **contextual observer**. At this stage, the project mirrors autonomy, except it is deployed at a doorway instead of a drone.

## BACKEND AND DATABASE STACK

Iteration one barely acknowledges the need for a backend; logs may be serial prints, and user data lives in a struct or JSON file on the MCU.

Iteration two creates a real backend—FastAPI, Flask, or Express—exposing endpoints for logs, user enrolment, and configuration. SQLite or PostgreSQL stores embeddings and access attempts. Passwords are hashed properly. Logs can be retrieved. You have observability.

Iteration three introduces structure and scale. Device registration, role-based administration, multi-door support,

and cached sessions appear. PostgreSQL becomes standard; Redis might join. The backend becomes authoritative, and the lock trusts but verifies. Remote updates and configuration sync become normal. The database schema no longer resembles a demo—it resembles something with migrations and schema guarantees.

Iteration four moves from competent to correct. Logs may become append-only or signed to resist tampering. Firmware updates can be attested. Backend services gain staging and production environments. Nothing here is experimental anymore.

## FRONTEND STACK

Iteration one has no frontend. Your interface is UART printouts and prayer.

Iteration two introduces a basic dashboard—perhaps React or Vue—allowing admins to enrol users, view logs, and adjust lock timings.

Iteration three transforms the dashboard into a management console. Multiple locks, multiple sites, reporting, charts, user groups, and RBAC appear. The interface is designed, not hacked.

Iteration four refines rather than explodes. The frontend becomes polished, tested, and governed by product sensibilities rather than engineering whims.

## SECURITY AND CI/CD STACK

Iteration one ignores security. Anyone with physical access owns the system. Iteration two adds HTTPS, hashed passwords, and sane defaults. Iteration three introduces secure identity, rate limits, OTA software delivery, and optional hardware trust anchors. CI/CD tests behavioural flows. Iteration four completes the arc: signed firmware, secure boot, audit-integrity guarantees, reproducible builds, and code scanning. At this point, the system isn’t just secure—it has a *security posture*.

Once the secure enclave is added, software ceases to be the root of trust. Cryptographic identity lives in silicon. Firmware updates must be signed. Unlock events are attested and stored immutably. If the enclosure is breached, keys self-destruct. Replay attacks become meaningless; cloned devices refuse to operate without cryptographically verifiable provenance. The door now defends itself. Physical tamper-resistance must extend beyond intrusion detection and cryptographic vaulting. Debug interfaces such as UART, JTAG, and SWD must be permanently fused, epoxy-shielded, or electrically isolated to prevent adversaries from probing internal states or extracting firmware. For high-assurance environments, anti-decapping strategies and conductive shielding can deter invasive microprobing attacks. Without such protections, even the most rigorous software security measures become irrelevant, as attackers can bypass logic by directly interrogating the hardware substrate.

When behavioural analytics join, the lock evolves from protecting space to protecting *context*. It refuses unlocks at

impossible times, raises flags on anomalous behaviour, and resists coercion by evaluating semantics instead of inputs. This is the line between IoT junk and **security infrastructure**.

Also, while the current liveness mechanisms confirm that a face presented to the system corresponds to a physical human, but they do not yet address attacks designed to deceive the recognition pipeline directly. Robust biometric systems incorporate adversarial ML defences, including print and replay-attack mitigation, mask and prosthetic detection, depth or IR-based verification, CNN perturbation pattern analysis, and temporal facial-consistency evaluation. These defences harden the model against synthetic or manipulated stimuli, ensuring that the recognition subsystem remains trustworthy even when confronted with intentional deception rather than simple spoofing. This elevates the lock's trust model from passive verification to active adversarial resistance, which is essential when biometrics become the primary authentication factor.

In addition to encrypted credential storage, a secure biometric system must implement a defined credential lifecycle to maintain long-term trustworthiness. This includes key rotation policies to limit exposure in the event of compromise, revocation protocols to invalidate previously enrolled identities, and mechanisms to mitigate biometric drift caused by natural changes in a user's appearance over time. Without these operational policies, credential data becomes static and vulnerable, undermining the integrity of the authentication pipeline even when stored securely.

While the current CI/CD pipeline enforces secure builds and signed firmware images, real-world update workflows require staged deployments, rollback paths, and diff-based firmware distribution. Staged rollouts prevent untested updates from simultaneously bricking multiple devices, while rollback protection ensures that compromised firmware cannot be reinstated. Diff-based delivery reduces bandwidth and minimises update time, especially in offline or low-connectivity deployments. Together, these mechanisms transform firmware updates from a development convenience into a controlled, resilient operational process.

### THREAT MODELLING

A comprehensive threat model extends beyond authentication correctness and evaluates the system's resilience against intentional deception and hardware manipulation. Attack vectors include replay attacks using printed or displayed facial imagery, adversarial perturbations crafted to mislead convolutional models, firmware tampering through unsigned update paths, and RF or power-glitching techniques capable of inducing unintended state transitions. Physical intrusions—such as drilling into the enclosure or probing debug pins—pose direct extraction risks, while social threats include coercion, identity theft, or presentation of stolen biometric samples. To reason about these hazards systematically, a STRIDE or DREAD-based categorisation can be applied, mapping each identified

vector to mitigation strategies and residual risk levels. Moreover, incorporating an adversarial ML defence taxonomy—covering spoof-resistant cues, distribution-shift monitoring, and perturbation-aware inference—ensures the recognition pipeline remains trustworthy even against adaptive attackers. This elevates the lock from a reactive controller to a secure, adversarially aware cyber-physical system.

### CONCLUSION

Unlike existing commercial smart locks, which rely heavily on cloud-based authentication pipelines, user-held tokens, and shallow liveness checks, the proposed architecture executes biometric inference, credential evaluation, and actuation logic entirely at the edge. Commercial platforms typically lack hardware trust anchors, multimodal biometric fusion, adversarial ML defences, and deterministic control behaviour—resulting in systems that are trivial to spoof with printed photographs or replayed imagery. By integrating secure enclaves, temporal identity models, behavioural analytics, and optional hardware acceleration, this platform shifts from a convenience appliance to a verifiable security infrastructure. In effect, the system embodies features that incumbent vendors market aspirationally but do not implement, positioning it not as an incremental improvement but as a categorical redefinition of smart access control.

### REFERENCES

- [1] "AI-Powered Face Recognition Smart Lock System with RFID and IoT Backup," Arduino Forum, Apr. 14, 2025. [Online]. Available: <https://forum.arduino.cc/t/ai-powered-face-recognition-smart-lock-system-with-rfid-and-iot-backup/1373376>. [Accessed: 05-Dec-2025].
- [2] A. Choudhary, "ESP32-CAM Face Recognition Door Lock System," CircuitDigest, Aug. 19, 2024. [Online]. Available: <https://circuitdigest.com/microcontroller-projects/esp32-camface-recognition-door-lock-system>. [Accessed: 05-Dec-2025].
- [3] Y. Yogeshwaran, "How to Make an RFID Door Lock System using Arduino?" CircuitDigest, Feb. 19, 2025. [Online]. Available: <https://circuitdigest.com/microcontroller-projects/how-to-make-an-rfid-door-lock-system-using-arduino>. [Accessed: 05-Dec-2025].