**Hackathon Report: Personal AI Assistant with Secure Face Verification**

**Executive Summary**

Our project introduces a **Personal AI Assistant** secured by a **CNN-powered face-recognition login system**. Users authenticate by presenting a face image (live or uploaded), which is matched against stored templates linked to their username. The system either admits the user or denies access in case of a mismatch, with all failed attempts logged and escalated.

Authenticated users are granted access to their personal AI assistant, powered by a **centralized server-side database**. This architecture reduces client-side computation, improves scalability, and enables **cross-domain insights** into health, finance, and scheduling.

The assistant provides **real-time, voice-enabled interaction**, delivering analytics and recommendations by linking financial, health, and calendar data. Example correlations include:

* High stress scores paired with tightly packed meetings.
* Elevated heart rate averages on days with high spending and outdoor activities.

**Face Verification at Low Latency (Technical Details)**

**Task Definition**

**Problem:** Verify if a given query image belongs to a claimed identity (1:1 verification).  
**Constraints:**

* Dataset per user: 100–1500 images, including duplicates and low-quality samples.
* Latency: ≤500 ms end-to-end (CPU baseline), <50 ms target (GPU).
* Security: Target **False Accept Rate (FAR) ≤ 0.1%**.

**Baseline vs Optimized Pipeline**

**Baseline:**

* Extract 128-D embeddings for each image (dlib/face\_recognition).
* Compare query embedding to *all* stored embeddings for a user.
* Threshold on raw distances.

⚠ Issues: High latency (hundreds of comparisons per verification), noisy thresholds, and redundancy from duplicate images.

**Final Pipeline (Optimized):**

1. **Preprocessing**
   * Detect faces with HOG (CPU) or CNN (GPU for higher accuracy).
   * Apply quality gates:
     + **Blur filter** via Laplacian variance (threshold ≥40).
     + Minimum bounding box size ≥64 px.
   * Deduplicate embeddings (cosine similarity ≥0.995 treated as duplicates).
2. **Embedding**
   * Use 128-D L2-normalized vectors (standard in biometrics).
   * Embedding latency dominates pipeline (~350–400 ms CPU).
3. **Template Generation (Offline Enrollment)**
   * Run **K-Means clustering** on embeddings per user.
   * Choose **K = min(5, floor(N/12))**, yielding typically 3–8 templates.
   * Persist templates in .npy format (few kilobytes per user).
4. **Verification (Online)**
   * Embed query face → vector q.
   * Load stored templates T ∈ ℝ^{K×128}.
   * Compute cosine similarity:

s=max⁡i=1..K⟨q,Ti⟩s = \max\_{i=1..K} \langle q, T\_i \rangles=i=1..Kmax​⟨q,Ti​⟩

* + Accept if **s ≥ τ**, where τ is pre-calibrated.
  + Complexity: O(KD) (≈0.01 ms on CPU for K≤8).

1. **Threshold Calibration**
   * Run held-out test splits (70/30).
   * Evaluate **TPR, FPR, EER (Equal Error Rate), and Best-F1 thresholds**.
   * Select τ at target FAR ≈0.1% → ensures high-security operation.

**Performance Results**

* **Two-user sanity test**:
  + EER=0.0, Best-F1=1.0.
  + TPR=100%, FPR=0%.
  + Scoring latency: 0.06 ms; embedding latency: ~353 ms (CPU).
* **50-user stress test**:
  + Threshold τ ≈ 0.898 → TPR=100%, FPR≈0.098% (≈4 false accepts in 4,067 impostor trials).
  + Embedding latency: p50=390 ms, p95=439 ms (CPU).
  + Scoring latency: p50=0.01 ms.

**Why K-Templates Improve Stability**

* **Single centroid limitation:** Mean embedding collapses pose/lighting clusters → reduces genuine similarity.
* **K-Means centroids:** Capture multiple modes (frontal/profile, glasses/no glasses, lighting variations).
* At verification: **max cosine** picks the best-matching mode → preserves genuine score while suppressing impostor score.
* Reduction from 100–150 embeddings → 3–8 templates improves speed and stabilizes thresholds.

**Admin Portal (Built from Scratch)**

We developed a **custom admin dashboard** to make the system usable and secure in real-world deployments:

* **User Management**
  + Add/update users by uploading image folders.
  + Option to overwrite existing templates.
  + Manage 50+ registered identities.
* **Log Monitoring**
  + Dashboard view of **successful recognitions, failed attempts, and daily totals**.
  + Record metadata: **username, IP address, timestamp, accuracy score**.
  + Visual accuracy indicators and logs exportable for auditing.

This ensures administrators have full **control, transparency, and accountability** in managing access.

**Limitations & Future Work**

* Embedding is the primary bottleneck; GPU optimization or mobile-optimized CNN models (e.g., MobileFaceNet) will reduce latency.
* Add **liveness detection** to defend against spoofing attacks (blink detection, IR/depth checks).
* Extend clustering to adaptive enrollment, updating templates as user images evolve.

**Conclusion**

We engineered a **low-latency, high-accuracy face verification pipeline** and integrated it with a **personal AI assistant** that links health, finance, and scheduling insights. Our **scratch-built admin panel** empowers secure user registration and real-time log monitoring.

This system balances **biometric rigor** with **life-enhancing AI insights**, positioning it as a strong, deployment-ready hackathon solution.