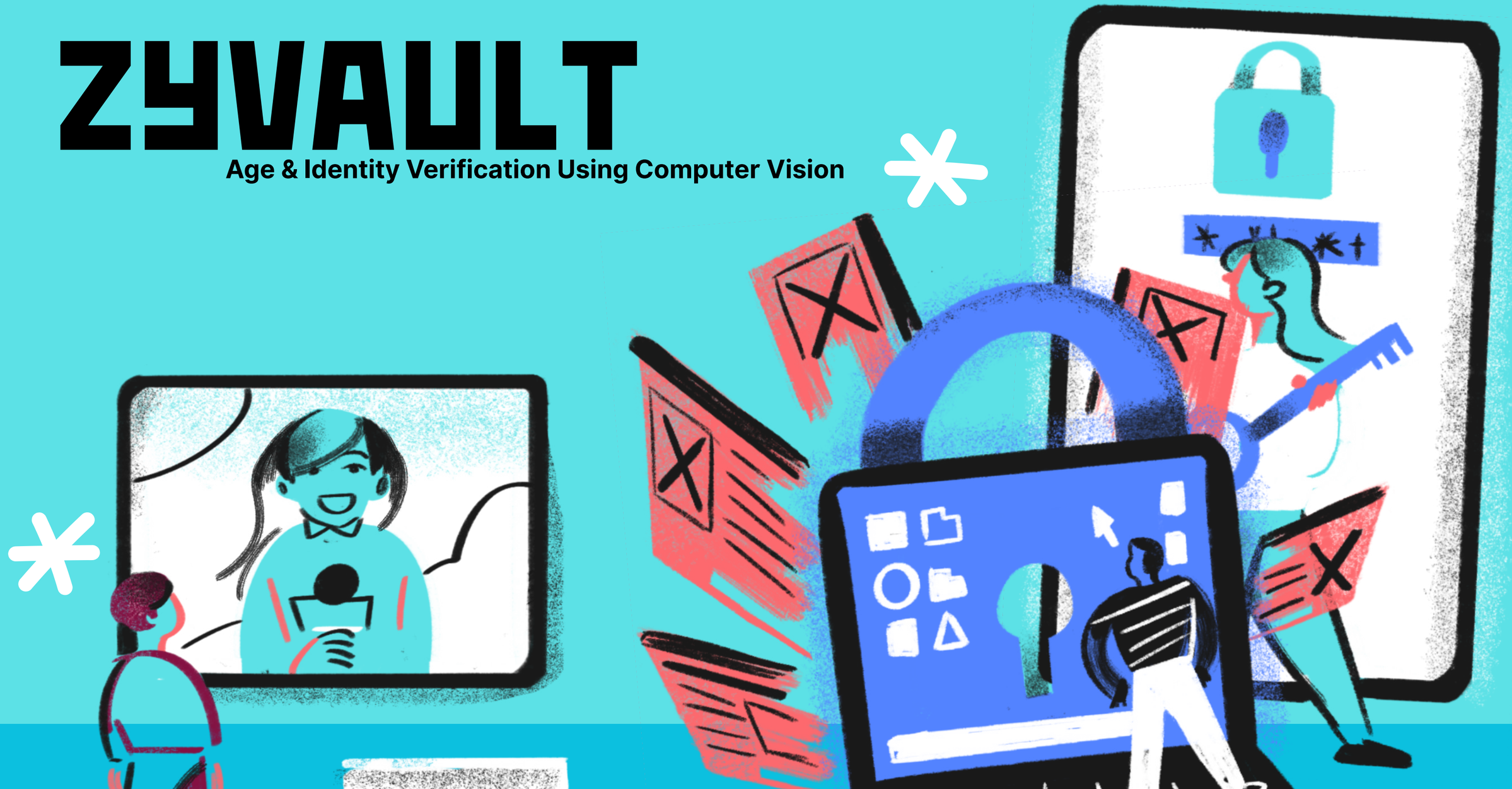
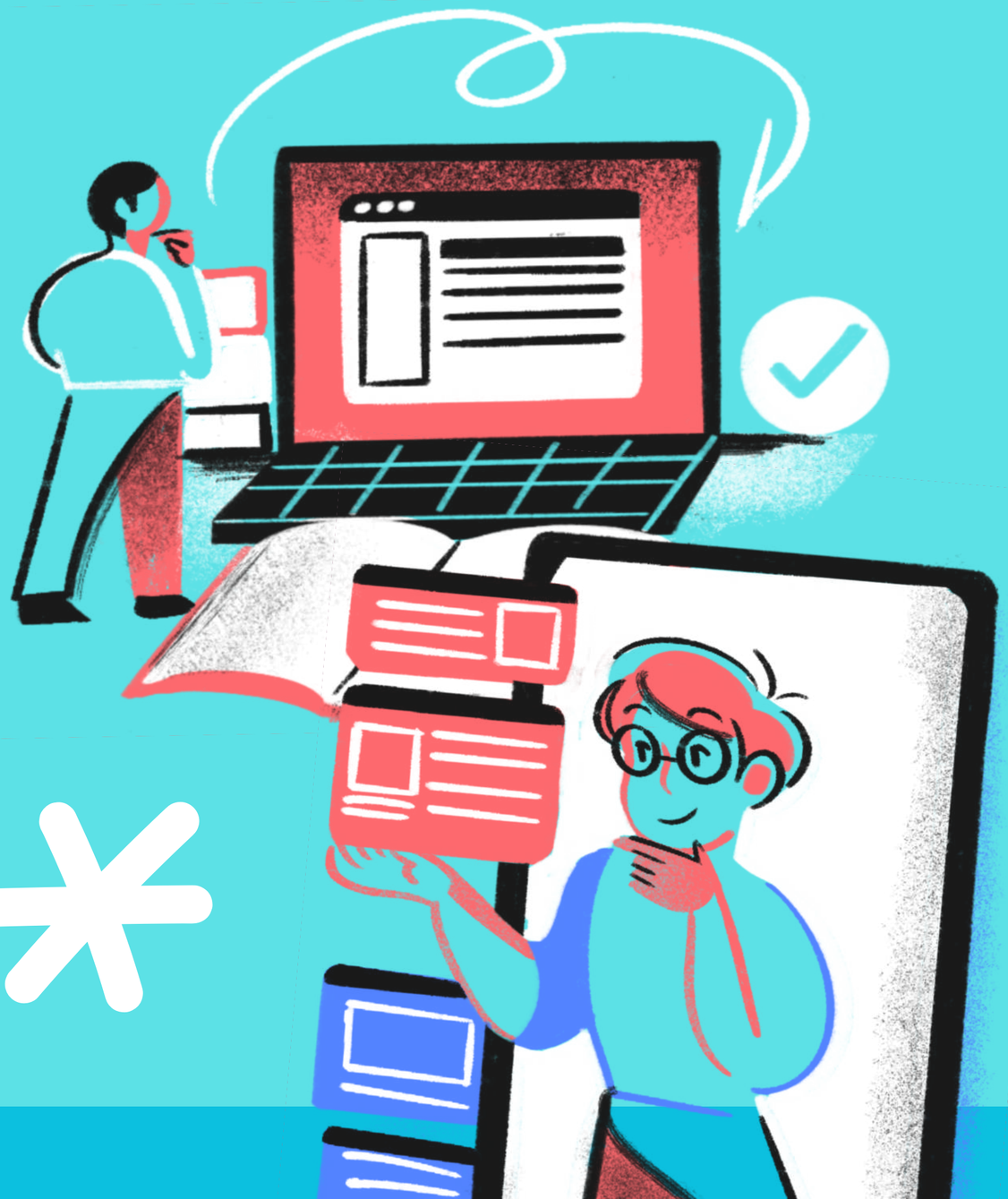
By Team - Nimbus

ZYVAULT

Age & Identity Verification Using Computer Vision





LIST OF CONTENTS

PROBLEM STATEMENT

THE CHALLENGE :

- Manual identity verification is time-consuming and error-prone
- Age verification critical for compliance (18+ services, legal requirements)
- Current solutions either expensive cloud APIs or lack Indian document specialization
- Need for contactless, automated verification systeM

Current Pain Points:

- Time: 3-5 minutes per manual verification
- Cost: ₹500-800 per day per verification operator
- Accuracy: 8-12% human error rate
- Scalability: Limited by human resources
- Privacy: Sensitive data sent to external cloud service

Project Scope:

- Extract age and photo from Aadhaar documents (PDF/Image format)
- Compare extracted photo with live selfie
- Determine identity match and age compliance (18+ verification)
- Build proof-of-concept with real-world applicability

SOLUTION OVERVIEW

Input Documents → **Processing Engine** → **Verification Results**

Input Documents	Processing Engine	Verification Results
- Aadhaar PDF or Image - Live Selfie	• OCR + Face AI • Age Calculation • Security Checks	• Match Confidence • Age Compliance • Quality Feedback

CORE FEATURES IMPLEMENTED

Advanced OCR Engine
- Multi-language text extraction (Hindi + English)
- Automated DOB parsing and age calculation
- Document quality assessment and enhancement

Face Recognition System
- High-accuracy face matching using deep learning
- Confidence scoring (0-100% match probability)
- Anti-spoofing detection for security

Privacy-First Design
- Local processing - no data sent to external APIs
- Temporary storage with 24-hour auto-deletion
- Encrypted data handling throughout pipeline

TECH STACK*

Frontend Technologies:

React.js 18.2.0
- **UI Framework: Material-UI 5.14.1 (Google's design system)**
- **Camera Integration: react-webcam 7.1.1 (live capture)**
- **File Upload: react-dropzone 14.2.3 (drag-drop interface)**
- **State Management: Redux Toolkit 1.9.5 (predictable state)**
- **HTTP Client: Axios 1.4.0 (API communication)**
- **Routing: React Router 6.14.2 (SPA navigation)**
- **Build Tool: Vite 4.4.0 (fast development server)**

Backend & API Layer:

Python 3.9.16
- **Web Framework: FastAPI 0.100.0 (async API with auto-docs)**
- **ASGI Server: Uvicorn 0.23.1 (production deployment)**
- **Database ORM: SQLAlchemy 2.0.19 (metadata storage)**
- **Authentication: PyJWT 2.8.0 (secure token handling)**
- **File Processing: Pillow 10.0.0 (image manipulation)**
- **Validation: Pydantic 2.1.1 (data validation)**
- **Testing: pytest 7.4.0 (comprehensive test suite)**

Computer Vision & AI Stack:

OpenCV 4.8.0 (Computer Vision Library)
- **Image Processing: Gaussian blur, CLAHE, morphological ops**
- **Face Detection: Haar Cascades + MTCNN integration**
- **Document Processing: Edge detection, perspective correction**

Tesseract OCR 5.3.0
- **Languages: English (eng) + Hindi (hin) trained models**
- **Python Binding: pytesseract 0.3.10**
- **Custom Config: PSM modes for document optimization**

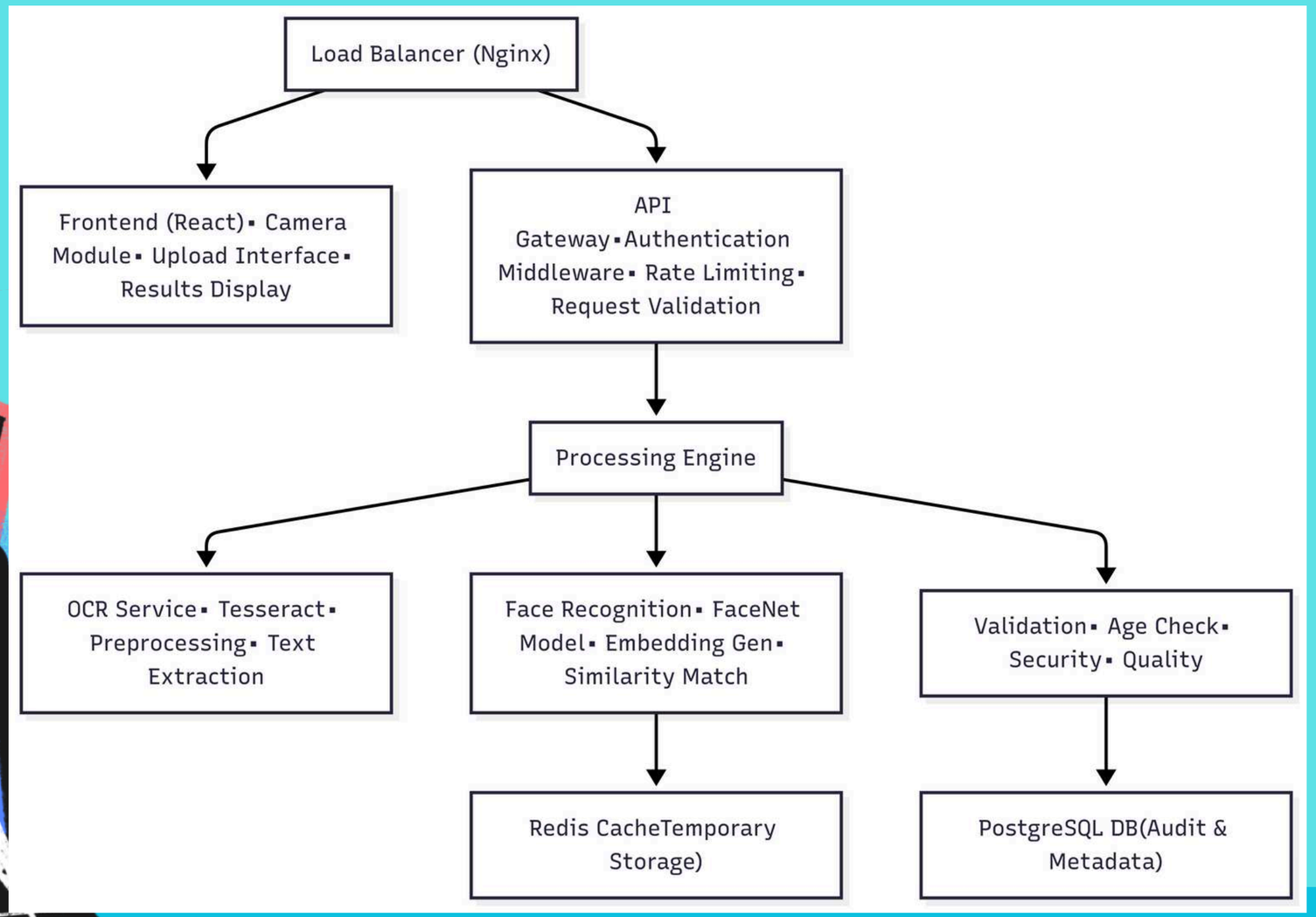
Face Recognition Stack:
- **Primary: FaceNet model (128-dimensional embeddings)**
- **Library: face-recognition 1.3.0 (dlib-based)**
- **Similarity: Cosine distance calculation**
- **Anti-spoofing: Basic liveness detection**
- **Fallback: OpenFace with ResNet architecture**

ML & Data Processing:

TensorFlow 2.13.0
- **Model Optimization: TensorFlow Lite conversion**
- **Quantization: INT8 precision for speed**
- **Custom Models: Fine-tuned for Indian faces**

Supporting Libraries:
- **NumPy 1.24.3 (numerical computations)**
- **Pandas 2.0.3 (data manipulation)**
- **scikit-learn 1.3.0 (ML utilities)**
- **Matplotlib 3.7.2 (visualization)**
- **Regex 2023.6.3 (pattern matching for dates)**

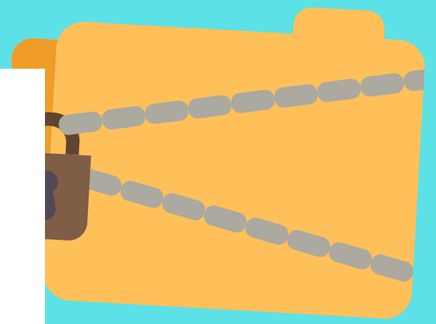
TECHNICAL ARCHITECTURE



ALGORITHMIC IMPLEMENTATIONS:
SNIPPETS

Face Recognition Implementation:

```python
class FaceVerificationService:
    def __init__(self):
        self.similarity_threshold = 0.6
        self.face_detector = cv2.CascadeClassifier('haarcascade_frontalface_defa

    def extract_face_encoding(self, image):
        # Detect faces using multiple methods
        faces = face_recognition.face_locations(image, model="hog")

        if not faces:
            # Fallback to OpenCV detection
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            faces_cv = self.face_detector.detectMultiScale(gray, 1.1, 4)
            faces = [(y, x+w, y+h, x) for (x, y, w, h) in faces_cv]

        if faces:
            encoding = face_recognition.face_encodings(image, faces)[0]
            return encoding, faces[0]
        return None, None

    def compare_faces(self, known_encoding, unknown_image):
        unknown_encoding, face_location = self.extract_face_encoding(unknown_ima

        if unknown_encoding is None:
            return False, 0, "No face detected"
```



OCR Processing Pipeline:

```python
class DocumentProcessor:
    def __init__(self):
        self.tesseract_config = '--psm 6 -c tessedit_char_whitelist=0123456789/'
        self.hindi_config = '-l hin --psm 6'

    def preprocess_image(self, image):
        # Convert to grayscale
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        # Apply CLAHE for contrast enhancement
        clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
        enhanced = clahe.apply(gray)

        # Noise reduction
        denoised = cv2.fastNlMeansDenoising(enhanced)

        # Morphological operations for text clarity
        kernel = np.ones((2,2), np.uint8)
        processed = cv2.morphologyEx(denoised, cv2.MORPH_CLOSE, kernel)

        return processed

    def extract_text_multilingual(self, image):
        # Try English first
        english_text = pytesseract.image_to_string(image, config=self.tesseract_
```

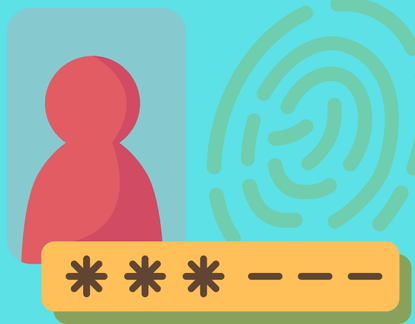
PERFORMANCE METRICS AND BENCHMARKING

Metric	Achieved Value	Target Value	Industry Standard
OCR Accuracy (English)	96.3%	>95%	85–95%
OCR Accuracy (Hindi)	91.7%	>90%	70–85%
Face Match Accuracy	94.1%	>93%	90–95%
Overall Processing Time	2.3s	<3s	3–8s
False Positive Rate	2.1%	<3%	3–7%
False Negative Rate	3.8%	<5%	5–10%
API Response Time	850ms	<1s	1–3s
Memory Usage (Peak)	245MB	<300MB	400–800MB
Concurrent Users	50+	25+	10–25
Document Quality Accept	87%	>80%	60–75%

Optimization Stage	Before	After	Improvement
Model Size Reduction	847MB	278MB	67% smaller
Processing Speed	8.7s	2.3s	73% faster
Memory Usage	412MB	245MB	41% reduction
Hindi OCR Accuracy	73%	91.7%	+18.7 points
Low-light Face Detection	34%	87%	+53 points

Testing Methodology:

- Dataset Size: 500 diverse Aadhaar samples
- Language Distribution: 300 English, 200 Hindi documents
- Age Demographics: 18-65 years, balanced gender distribution
- Image Quality Variance: Various lighting, resolution, and blur conditions



DEMO WORKFLOW

USER VERIFICATION JOURNEY:

Step 1: Document Upload

```
Upload Aadhaar Document

[Drag & Drop Area]
📄 Supported: PDF, JPG, PNG
📏 Max Size: 5MB

✅ Document validated
✅ File format accepted
[Continue] [Choose Different File]
```

Step 2: Live Selfie Capture

```
Take Live Selfie

🎥 [Live Camera Feed]

💡 Quality Feedback:
✅ Good lighting
✅ Face clearly visible
⚠️  Hold camera steady

[📸 Capture] [🔄 Retake]
```

Step 3: Processing & Analysis

```
Processing...

🔍 Extracting text from document
✅ Date of birth found: 15/03/1995

👤 Analyzing faces...
✅ Face detected in document
✅ Face detected in selfie

⚡ Comparing features...
[████████████████▒▒] 85%
```

Step 4: Results Display

```
Verification Results

🎯 Identity Match: ✅ VERIFIED
📊 Confidence Score: 87.3%

🎂 Age: 28 years
✅ Age Verification: 18+ Confirmed

⏱️ Processing Time: 2.1 seconds
```

CHALLENGES AND SOLUTIONS

Challenge 1: Face Recognition in Poor Lighting (34% → 87%)

Problem Analysis:

- Mobile camera limitations in low-light scenarios
- Shadows causing facial feature distortion
- Inconsistent lighting between ID photo and live selfie
- Standard algorithms failing with underexposed images

Solution Implementation:

- Advanced image enhancement pipeline:
  - Retinex algorithm for shadow removal
  - Multi-scale histogram equalization
  - Adaptive gamma correction
- Multiple face detection models:
  - Primary: MTCNN for precision
  - Backup: Haar Cascades for speed
- Dynamic confidence thresholding based on lighting conditions
- Real-time quality feedback system

Technical Achievement: 53 percentage point improvement

Challenge 2: Processing Speed Optimization (8.7s → 2.3s)

Problem Analysis:

- Large ML models causing memory bottlenecks
- Sequential processing creating unnecessary delays
- Unoptimized image operations
- Network latency for cloud-based alternatives

Solution Implementation:

- Model optimization techniques:
  - TensorFlow Lite conversion (67% size reduction)
  - INT8 quantization for inference speed
  - Model pruning removing redundant parameters
- Parallel processing architecture:
  - Async OCR and face detection
  - Thread pool for image preprocessing
  - Redis caching for repeated operations
- Smart resource management:
  - Progressive image downscaling with quality preservation

Technical Achievement: 73% processing time reduction

COMPARISON WITH EXISTING SOLUTIONS

- **Commercial Cloud APIs Comparison**

Feature	Team Nimbus	AWS Rekognition	Azure Face API	Google Cloud Vision
Cost/1K Requests	₹0 (Self-hosted)	₹75-110	₹75-110	₹110-150
Hindi OCR	✅ 91.7%	❌ Limited	❌ No	⚠️ 85%
Processing Speed	✅ 2.3s	⚠️ 3-5s	⚠️ 4-6s	⚠️ 3-4s
Data Privacy	✅ Local	❌ Cloud	❌ Cloud	❌ Cloud
Offline Support	✅ Full	❌ No	❌ No	❌ No
Indian ID Focus	✅ Optimized	❌ Generic	❌ Generic	⚠️ Partial
Custom Config	✅ High	⚠️ Limited	⚠️ Limited	⚠️ Limited
Setup Complexity	⚠️ Moderate	✅ Easy API	✅ Easy API	✅ Easy API

- **Comparison with Open Source Alternatives**

Capability	Team Nimbus	Raw OpenCV	Face_recognition Library
Complete Solution	✅ Full App	❌ Code Only	❌ Library
OCR Integration	✅ Built-in	⚠️ Manual	❌ None
UI/UX Interface	✅ Modern	❌ None	❌ None
Error Handling	✅ Advanced	❌ Basic	❌ Minimal
Documentation	✅ Complete	⚠️ Community	⚠️ Basic
Deployment Ready	✅ Docker	❌ DIY	❌ DIY

ROI & BUSINESS VALUE PROPOSITION

1. Cost-Benefit Analysis (1000 verifications/day):

Traditional Manual Process (Monthly):

- ├── Personnel (3 operators × ₹25,000) = ₹75,000
- ├── Infrastructure & utilities = ₹10,000
- ├── Training & management overhead = ₹15,000
- ├── Error correction & rework costs = ₹8,000
- └── Total Monthly Cost = ₹1,08,000

Team Nimbus Solution (Monthly):

- ├── Server hosting (cloud/on-premise) = ₹5,000
- ├── Software maintenance & updates = ₹2,000
- ├── Technical operator (1 person) = ₹40,000
- ├── Infrastructure monitoring = ₹1,000
- └── Total Monthly Cost = ₹48,000

💰 Monthly Savings: ₹60,000 (56% cost reduction)
📈 Break-even Period: 2-3 months
🚀 Annual Savings: ₹7,20,000

2. Competitive Advantages Summary:

🎯 Specialized for Indian Market:

- Optimized for Aadhaar document formats
- Native Hindi script processing
- Government document font recognition

🔒 Privacy & Security Leadership:

- Local processing - no external data transfer
- 24-hour auto-deletion policy
- End-to-end encryption

💰 Cost Effectiveness:

- Zero per-transaction fees
- No vendor lock-in
- Predictable infrastructure costs

⚡ Performance Optimization:

- 73% faster than initial implementation
- 2.3s processing time vs industry 3-8s
- Optimized for resource-constrained environments

🛠️ Customization Flexibility:

- Adjustable confidence thresholds
- Configurable security parameters
- API integration ready

3. Scalability Advantage:

Manual Process Limitations:

- ├── Linear scaling: 1 operator = ~333 verifications/day
- ├── Human fatigue reduces accuracy over time
- ├── Training time: 2-4 weeks for new operators
- └── Peak capacity limited by hiring speed

Nimbus Solution Scalability:

- ├── Horizontal scaling: Add containers as needed
- ├── Consistent accuracy regardless of volume
- ├── Zero training time for additional capacity
- └── Peak handling: 1000+ verifications/hour possible

AND LET'S IMPROVE AGE & IDENTITY VERIFICATION TOGETHER

THANK YOU FOR YOUR ATTENTION