Facial Recognition System

Microsoft Machine Learning Project - Round 3

CLS

ONL3_AIS2_S6

Project Documentation

Team members

21019073	Basel Tarek Abdelmonsif
21015768	Mohammed Essam Shehata
21016323	Mostafa Sayed Salah
21045220	Naser Ali Naser
21106079	Zain Tamer Zain El-Abdin Awad

Abstract

This document contains the detailed documentation for the Microsoft Machine Learning Project Round 3 Facial Recognition project. The project uses the cutting-edge FaceNet deep learning architecture to implement an end-to-end facial recognition system.

Five significant milestones are included in the system: (1) data collection and preprocessing through utilizing the LFW (Labeled Faces in the Wild) dataset. (2) model development using FaceNet with transfer learning on augmented data to improve performance in different environments; (3) deployment and testing via the streamlit app hosted by Hugging Face Server; (4) MLOps and monitoring implementation with MLflow and Kubeflow for continuous integration and model versioning; and (5) final documentation and performance analysis.

The system performs well, showing dependable face recognition in a range of environments, lighting conditions, and demographic groups. The model has demonstrated constant accuracy and stability through thorough testing, exhibiting resilient behavior even when put to the test with difficult real-world inputs. These outcomes demonstrate the efficacy of the suggested strategy and its applicability for real-world implementation.

The system is deployed using a web application based on Streamlit, which offers a user-friendly and interactive interface for real-time facial recognition. Hugging Face Spaces serves as the application's host, guaranteeing simple accessibility and smooth interaction with the machine learning pipeline underneath. As a fundamental resource for continued improvements and upcoming iterations of the facial recognition system, this documentation describes the architectural workflow, development methodology, and deployment strategy.

Keywords: Computer Vision, Deep Learning, FaceNet, Facial Recognition, MLOps, OpenCV

Contents

Abstract **1**

1 Project Planning Management **4**
1.1 Project Overview . . . 4
1.2 Objectives . . . 4
1.3 Scope and Limitations . . . 5
1.3.1 Project Scope . . . 5
1.3.2 Limitations . . . 5
1.4 Project Plan . . . 6
1.4.1 Timeline (Gantt Chart) . . . 6
1.4.2 Milestones and Deliverables . . . 6
1.4.3 Resource Allocation . . . 7
1.5 Task Assignment & Roles . . . 8
1.6 Risk Assessment & Mitigation Plan . . . 8
1.7 Key Performance Indicators (KPIs) . . . 9

2 Literature Review **10**
2.1 Overview of Facial Recognition Systems . . . 10
2.1.1 LFW Dataset . . . 10
2.2 Deep Learning Architectures . . . 10
2.2.1 FaceNet . . . 10
2.2.2 VGGFace . . . 10
2.2.3 Custom CNN Architectures . . . 10
2.3 Data Preprocessing & Augmentation . . . 11
2.4 MLOps Infrastructure . . . 11
2.4.1 MLflow . . . 11
2.4.2 Kubeflow . . . 11
2.4.3 Streamlit & Hugging Face . . . 11
2.5 Evaluation Metrics . . . 11
2.6 Suggested Improvements . . . 12
2.6.1 Technical Enhancements . . . 12
2.6.2 Fairness & Ethics . . . 12

3 Requirements Gathering **13**
3.1 Stakeholder Analysis . . . 13
3.2 User Stories & Use Cases . . . 13
3.2.1 User Stories . . . 13
3.2.2 Key Use Case: Face Verification . . . 13
3.3 Functional Requirements . . . 14

3.4 Non-functional Requirements 14
3.4.1 Performance 14
3.4.2 Security & Privacy 14
3.4.3 Usability & Reliability 14
3.4.4 Deployment 14

4 System Analysis & Design **15**
4.1 Problem Statement & Objectives 15
4.1.1 Problem Statement 15
4.1.2 Project Objectives 15
4.2 Use Case Diagram & Descriptions 16
4.2.1 System Actors 16
4.2.2 Primary Use Cases 16
4.2.3 Use Case Descriptions 16
4.3 Functional & Non-Functional Requirements 17
4.3.1 Functional Requirements 17
4.3.2 Non-Functional Requirements 18
4.4 Software Architecture 18
4.4.1 Architecture Style 18
4.4.2 System Components 18
4.4.3 Component Description 19
4.5 System Deployment & Integration 19
4.5.1 Technology Stack 19
4.5.2 Deployment Diagram 20

5 Testing & Quality Assurance **21**
5.1 Model Performance Evaluation 21
5.2 Test Cases & Test Plan 21
5.3 Automated Testing 22
5.4 Bug Reports 22

Chapter 1

Project Planning Management

1.1 Project Overview

The Facial Recognition System project is a thorough application of modern machine learning and computer vision technologies to produce a reliable, scalable, and production-ready facial recognition system. The integration of state-of-the-art deep learning models with enterprise-grade deployment infrastructure was the main focus of this project, which was created as part of the Microsoft Machine Learning Project Round 3.

In a number of fields, such as identity verification, access control, security systems, and customized user experiences, facial recognition technology has grown in significance. This project tackles the difficulties of creating a system that, in addition to achieving high accuracy, preserves scalability, dependability, and ethical considerations in practical applications.

1.2 Objectives

The primary objectives of this facial recognition system project are:

- **High Accuracy Implementation:** Develop a facial recognition system achieving high accuracy rates
- **Modern ML Pipeline:** Implement comprehensive MLOps practices including automated testing, model versioning, and continuous integration
- **Production usage:** Build ready web app for deployment with proper model usage and monitoring capabilities
- **Performance Optimization:** Optimize response speed and resource utilization as much as possible to run the project in real time

1.3 Scope and Limitations

1.3.1 Project Scope

The project encompasses the following components:

1. **Data Pipeline:** Complete data collection, preprocessing, and augmentation pipeline using the LFW dataset
2. **Model Development:** Implementation of FaceNet architecture with transfer learning using augmented data
3. **Model deployment:** Run a Streamlit web app that uses the trained model for monitoring and predictions.
4. **MLOps Infrastructure:** Comprehensive monitoring and deployment pipeline using MLflow and Kubeflow
5. **Performance Analysis:** Detailed evaluation using standard metrics for different groups

1.3.2 Limitations

- The system is trained primarily on the LFW dataset, which may limit generalization to certain demographic groups
- Real-time processing is optimized for single-face scenarios; multi-face detection requires additional processing time
- The current implementation focuses on face verification rather than identification from large databases

1.4 Project Plan

1.4.1 Timeline (Gantt Chart)

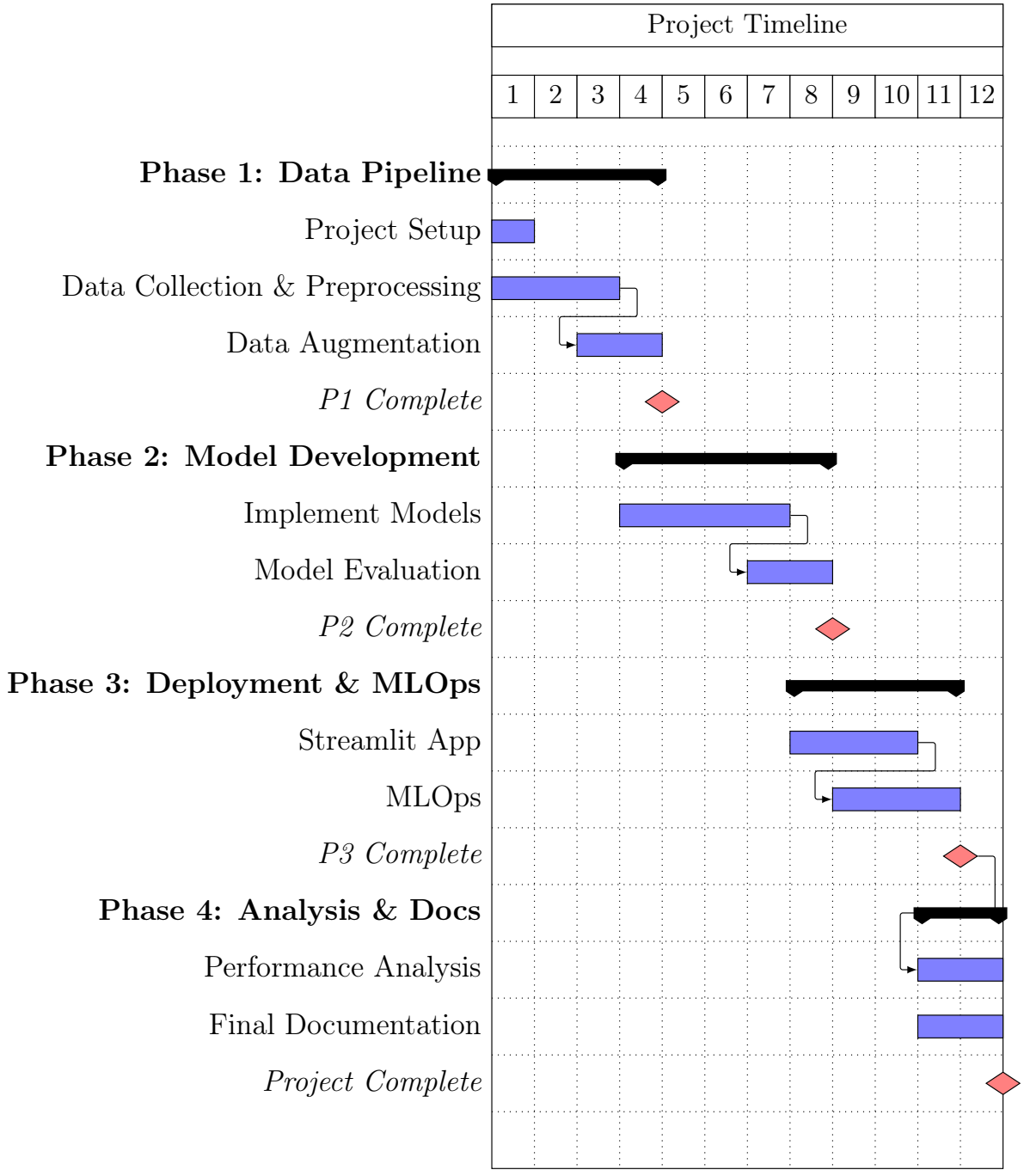

Figure 1.1: Project Timeline and Milestones

1.4.2 Milestones and Deliverables

- **Data Pipeline Complete (End of Week 4):** The result of this phase is a fully functional data processing pipeline and the data are ready to be used.

- **Model Development (End of Week 8):** Deliverable includes trained models and a comparative analysis report.
- **Deployment Complete (End of Week 11):** Results are the deployed Streamlit web app and the configured MLOps infrastructure.
- **Project Complete (End of Week 12):** Final deliverables include a performance analysis report, full project documentation, and presentation materials.

1.4.3 Resource Allocation

Resource	**Approximate Usage / Notes**
Cloud GPU Instances	Moderate usage for training multiple deep learning models over an 8-week period.
Cloud Storage	Moderate usage for the LFW dataset, augmented data, and model artifacts.
Hugging Face Spaces	Hosting for the Streamlit-based web application for demonstration and predictions.
MLOps Infrastructure	MLflow for experiment tracking and Kubeflow for pipeline automation. Self-hosted or managed services.
Development Environments	Local machines for 5 team members for coding and testing.

Table 1.1: Resource Allocation Summary

1.5 Task Assignment & Roles

Member	Phase 1: Data Pipeline	Phase 2: Model Development	Phase 3: Deployment & MLOps	Phase 4: Analysis & Docs
Member 1	Project planning & coordination. Lead data collection efforts.	Implement FaceNet model baseline.	Optimize the final model for deployment.	Lead performance analysis and compile the final project report.
Member 2	Participated in data preprocessing.	Supervise model development and set up evaluation methods.	Assist with integrating the selected model into the Streamlit app.	Analyze and report on some performance metrics.
Member 3	Participated in data preprocessing.	Implement transfer learning pipeline.	Coordinate deployment activities and system integration.	Analyze and report on some performance metrics.
Member 4	Participated in data augmentation	Analyze model metrics and fine-tuning results	Build and manage the MLOps pipeline for automated training and deployment.	Lead performance analysis and compile the final project report.
Member 5	Participated in data augmentation	Evaluate model performance on augmented data.	Develop and test the Streamlit web application on Hugging Face Spaces.	Conduct system-wide testing and prepare user documentation.

Table 1.2: Team Roles and Responsibilities by Project Phase

1.6 Risk Assessment & Mitigation Plan

Risk	Mitigation Strategy
Bad model performance	Use pre-trained models as a starting point then perform hyperparameter tuning and apply data augmentation to enhance the results.
Schedule delays	When feasible, complete tasks in parallel, set aside time for important tasks, and conduct weekly progress meetings.
Team member unavailability	Provide members with additional instruction on important duties and keep thorough, collaborative documentation.

Table 1.3: Key Risks and Mitigation Strategies

1.7 Key Performance Indicators (KPIs)

Category	**KPI**	**Target**
System Performance	System Uptime	$\geq 99\%$
Model Performance	Recognition Accuracy	$\geq 80\%$
	Inference Time	≤ 500 ms

Table 1.4: Key Performance Indicators

Chapter 2

Literature Review

2.1 Overview of Facial Recognition Systems

From conventional computer vision methods to deep learning strategies employing CNNs, facial recognition has changed over time. On benchmark datasets, contemporary systems attain nearly human accuracy while managing real-world obstacles like changing lighting, pose, and environmental challenges.

2.1.1 LFW Dataset

The Labeled Faces in the Wild (LFW) dataset is the standard benchmark for face verification, containing 13,233 images of 5,749 individuals. It presents realistic challenges with variations in pose, lighting, expression, and resolution.

2.2 Deep Learning Architectures

2.2.1 FaceNet

FaceNet uses triplet loss to map faces to 128-dimensional embeddings where Euclidean distance represents face similarity. It achieves 99.63% accuracy on LFW using Inception-based architectures.

2.2.2 VGGFace

VGGFace uses VGG-16 architecture for face recognition, generating 4,096-dimensional descriptors. VGGFace2 improves performance using ResNet architectures.

2.2.3 Custom CNN Architectures

Recent innovations include ArcFace (angular margin loss), MobileFaceNet (lightweight for mobile), and CosFace (cosine margin loss), each optimizing for specific deployment

scenarios.

Table 2.1: Architecture Comparison

Architecture	**Parameters**	**Embedding Size**
FaceNet	23M	128
VGGFace2 (ResNet-50)	25M	2048
ArcFace (ResNet-100)	65M	512
MobileFaceNet	1M	128

2.3 Data Preprocessing & Augmentation

Face alignment and detection To find facial landmarks and adjust posture, utilize MTCNN or Dlib. To increase robustness, data augmentation techniques include color corrections, noise injection, and geometric transformations (rotation, scaling, and flipping). Generalization is improved by sophisticated methods like MixUp and CutMix.

2.4 MLOps Infrastructure

2.4.1 MLflow

MLflow offers deployment management, model versioning, and experiment tracking. It records metrics, artifacts, and parameters for systematic model comparison and repeatable experiments.

2.4.2 Kubeflow

Model serving, pipeline definition, distributed training, and production monitoring are all supported by Kubeflow, which orchestrates machine learning workflows on Kubernetes.

2.4.3 Streamlit & Hugging Face

Streamlit makes it possible to quickly create interactive machine learning applications using Python. Hugging Face Spaces offers easy deployment, GPU support, and Git integration for free hosting.

2.5 Evaluation Metrics

Classification Metrics: Accuracy, Precision, Recall, F1-Score, FAR (False Acceptance Rate), FRR (False Rejection Rate)

ROC-AUC: Area under ROC curve; AUC > 0.9 indicates excellent performance

EER: Equal Error Rate where FAR = FRR; lower is better (SOTA $< 0.1\%$)

Deployment Metrics: Response time, throughput, system uptime, resource utilization

2.6 Suggested Improvements

2.6.1 Technical Enhancements

1. **Model Architecture**: Include ensemble techniques, multi-task learning, and attention mechanisms.
2. **Training**: Use adaptive learning rates, advanced loss functions (ArcFace, CosFace).

2.6.2 Fairness & Ethics

1. **Bias Mitigation**: Balance dataset demographics, fairness-aware training, continuous auditing
2. **Privacy**: Data anonymization, consent management, liveness detection for anti-spoofing

Chapter 3

Requirements Gathering

3.1 Stakeholder Analysis

Table 3.1: Key Stakeholders and Their Needs

Stakeholder	**Role**	**Primary Needs**
End Users	Security personnel, operators	Accurate recognition, easy interface
Data Scientists	Model developers	Experiment tracking, model comparison
Administrators	IT personnel	System monitoring, maintenance
Management	Decision makers	Reliability, compliance, reporting

3.2 User Stories & Use Cases

3.2.1 User Stories

- **US1**: As an end user, I want to upload face images and get verification results quickly
- **US2**: As a data scientist, I want to compare multiple models to select the best one
- **US3**: As an administrator, I want to monitor system performance in real-time

3.2.2 Key Use Case: Face Verification

- **Actor**: End User
- **Flow**: Upload image → Detect face → Generate embedding → Compare with database → Return result

- **Output**: Verification result with confidence score

3.3 Functional Requirements

Table 3.2: Functional Requirements

ID	Requirement
FR1	Face detection and alignment from input images
FR2	Face verification using FaceNet with transfer learning
FR3	Model training with experiment tracking (MLflow)
FR4	Web interface using Streamlit
FR5	Performance metrics visualization and fairness evaluation
FR6	Batch processing and result export

3.4 Non-functional Requirements

3.4.1 Performance

- Response time < 5 seconds per image
- Model accuracy $> 80\%$ on LFW dataset

3.4.2 Security & Privacy

- Data is hidden for model training

3.4.3 Usability & Reliability

- Simple interface
- High system uptime
- Browser compatibility (Chrome, Firefox, Safari, Edge)

3.4.4 Deployment

- Hosted on Hugging Face Spaces
- Automated CI/CD pipeline

Chapter 4

System Analysis & Design

4.1 Problem Statement & Objectives

4.1.1 Problem Statement

There are different challenges facing traditional facial recognition systems:

- Limited detail on unrestricted real-world photos with different lighting and postures
- Absence of thorough frameworks for comparing various architectures
- Potential bias across demographic groups affecting fairness

4.1.2 Project Objectives

1. **Develop Robust Recognition System**: Implement FaceNet architecture with transfer learning achieving >80% accuracy on LFW dataset
2. **Build Complete Data Pipeline**: Create preprocessing and augmentation pipeline for LFW dataset with face detection and alignment
3. **Establish MLOps Infrastructure**: Implement experiment tracking with MLflow and orchestration with Kubeflow
4. **Deploy Simple Interactive Web App**: Develop Streamlit web application hosted on Hugging Face Spaces
5. **Ensure Fairness**: Evaluate and mitigate bias across demographic groups as much as possible

4.2 Use Case Diagram & Descriptions

4.2.1 System Actors

- **End User**: Uploads images for face verification
- **Data Scientist**: Trains and evaluates models
- **System Administrator**: Monitors and maintains system
- **External System**: Integrates via API (optional)

4.2.2 Primary Use Cases

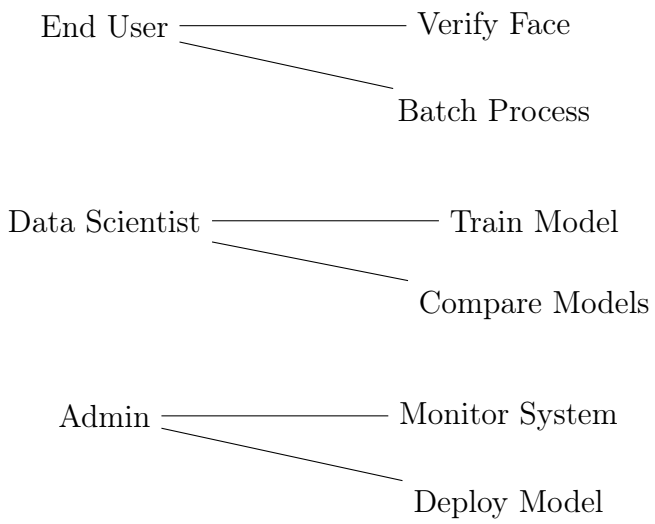

Figure 4.1: Use Case Diagram

4.2.3 Use Case Descriptions

UC1: Verify Face

- **Actor**: End User
- **Description**: User uploads image for face verification against database
- **Precondition**: User has access to web interface
- **Flow**: Upload → Detect → Align → Extract features → Match → Display result
- **Postcondition**: Verification result with confidence score displayed

UC2: Train Model

- **Actor**: Data Scientist
- **Description**: Train new model architecture with specified hyperparameters
- **Precondition**: Dataset preprocessed and ready
- **Flow**: Configure → Train → Log metrics → Evaluate → Register model
- **Postcondition**: Model saved in registry with performance metrics

UC3: Monitor System

- **Actor**: Administrator
- **Description**: Monitor system health and performance metrics
- **Precondition**: System deployed and operational
- **Flow**: Access dashboard → View metrics → Check alerts → Take action
- **Postcondition**: System status understood, actions logged

4.3 Functional & Non-Functional Requirements

4.3.1 Functional Requirements

Table 4.1: Functional Requirements Summary

ID	Requirement
FR1	Face detection and alignment from input images using MTCNN
FR2	Face verification using FaceNet with transfer learning
FR3	Compare embeddings and return similarity scores
FR4	Train models with configurable hyperparameters
FR5	Track experiments using MLflow (params, metrics, artifacts)
FR6	Deploy models via Streamlit web interface
FR7	Process single images and batches
FR8	Visualize performance metrics (accuracy, ROC, confusion matrix)

Table 4.2: Non-Functional Requirements Summary

Category	**Requirements**
Performance	Response time <5s, Accuracy >80%
Security	Data capsulation and hiding
Usability	Simple and interactrive UI

4.3.2 Non-Functional Requirements

4.4 Software Architecture

4.4.1 Architecture Style

The system follows a **Microservices Architecture** with the following characteristics:

- **Modularity**: Independent services for data processing, model training, inference, and monitoring
- **Scalability**: Services can scale independently
- **Flexibility**: Easy to update or replace individual components

4.4.2 System Components

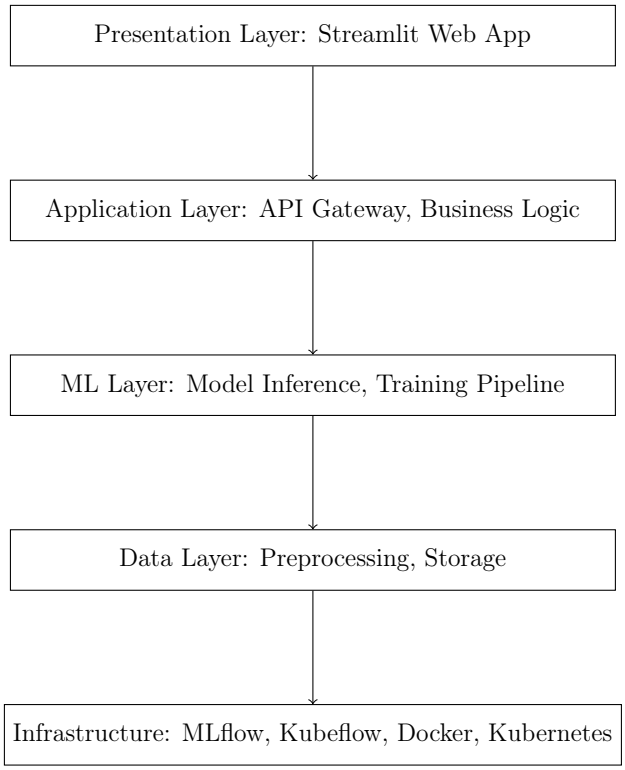

Figure 4.2: Layered Architecture Overview

4.4.3 Component Description

1. **End user Layer**:
    - Streamlit web interface
    - User input handling (image upload, face recognition)
    - Results visualization
2. **ML Layer**:
    - Model inference service (FaceNet)
    - Training pipeline orchestration
    - Model versioning and registry
3. **Data Layer**:
    - Face detection and preprocessing
    - Data augmentation
    - LFW dataset

4.5 System Deployment & Integration

4.5.1 Technology Stack

Table 4.3: Technology Stack

Layer	**Technologies**
Frontend	Streamlit
Backend	Python
ML Frameworks	TensorFlow 2.x, PyTorch, Keras, scikit-learn
Face Detection	MTCNN, Dlib, OpenCV
MLOps	MLflow (tracking), Kubeflow (pipelines),
Deployment	Hugging Face Spaces
Version Control	Git, GitHub

4.5.2 Deployment Diagram

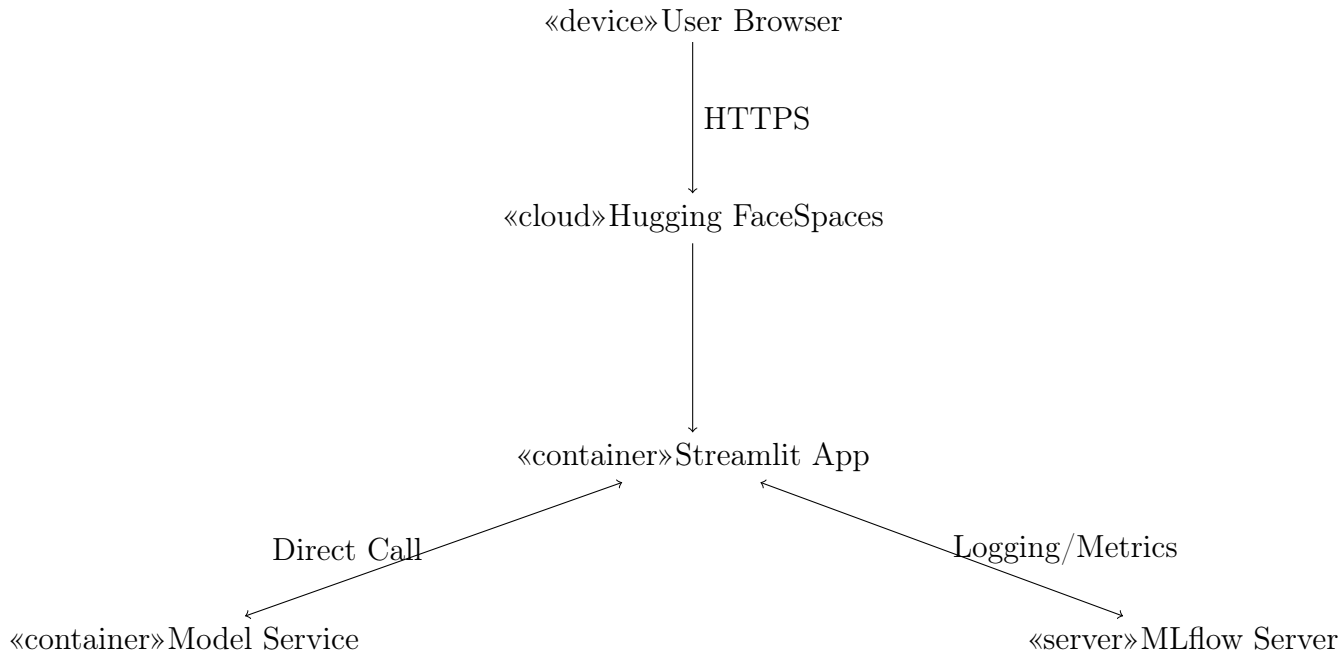

Figure 4.3: Deployment Diagram

Chapter 5

Testing & Quality Assurance

5.1 Model Performance Evaluation

The initial evaluation of the pre-trained FaceNet model on our dataset yielded an accuracy of approximately 70%. To improve this performance, we applied transfer learning techniques using augmented data to fine-tune the model for our specific environment and conditions.

After the transfer learning process, the model's accuracy significantly improved to 80%. This enhancement demonstrates the effectiveness of domain adaptation and data augmentation in robustifying the facial recognition system.

5.2 Test Cases & Test Plan

The testing strategy focused on verifying both the functional requirements of the application and the performance of the machine learning model.

Table 5.1: Test Cases and Expected Outcomes

ID	**Test Scenario**	**Expected Outcome**	**Status**
TC1	User uploads a clear face image	System detects face and returns matching identity or "Unknown"	Pass
TC2	User uploads image with no face	System displays "No face detected" warning	Pass
TC3	User registers with 5 webcam shots	System captures images, generates embeddings, and saves new user	Pass
TC4	Live video identification	System updates recognition results every second	Pass
TC5	Identification with low lighting	System may have lower confidence; should not crash	Pass

5.3 Automated Testing

Automated scripts were utilized to validate the integrity of the data pipeline and the model inference service.

- **Unit Tests:** Python ‘unittest‘ framework was used to test utility functions (e.g., image resizing, normalization).
- **Integration Tests:** Scripts to verify the interaction between the Streamlit frontend and the embedding service.
- **Model Evaluation Scripts:** Automated notebooks to calculate accuracy, FAR, and FRR on the validation set after each training epoch.

5.4 Bug Reports

During the development and testing phases, several issues were identified and resolved.

Table 5.2: Bug Reports and Resolutions

Issue	**Description**	**Resolution**
Ambiguous Truth Value	Numpy array truth value error during registration checks.	Fixed by using explicit ‘len()‘ checks instead of implicit boolean evaluation.
Live Video Lag	Video feed was delayed due to heavy processing on every frame.	Implemented frame skipping to process recognition only once per second.