National College of Ireland

Configuration Manual

Deepfake-Resistant Framework
for Real-Time Biometric Identity Verification in Remote Banking

Programme Name
MSC CyberSecurity

Shanthini Chandrasagaran
Student ID: x22197460

School of Computing
National College of Ireland

Supervisor: Ross Spelman

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Shanthini Chandrasagaran
Student ID:	X22197460
Programme:	MSC CyberSecurity **Year:** 2004
Module:	Configuration Manual
Lecturer:	Ross Spelman
Submission Due Date:	11th December 2025
Project Title:	Deepfake-Resistant Framework for Real-Time Biometric Identity Verification in Remote Banking.
Word Count:	2458 **Page Count:** 18

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Shanthini .....

Date: 08/12/2025 .....

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Shanthini Chandrasagaran
Student ID: x22197460

Introduction

This manual provides clear, step-by-step instructions to replicate the Deepfake-Resistant Framework. It outlines hardware/software requirements, dataset preparation, folder structures, configuration variables, and execution steps. The goal is to ensure reproducibility and clarity for academic review.

Core Components:

The framework integrates five biometric modalities:

- Facial recognition
- Voice authentication
- Behavioral biometrics
- Liveness detection
- Deepfake detection
- Vulnerability dashboard

2. Hard Requirements

The system was implemented on Google Colab GPU and a local Windows machine. Table below outlines the hardware configuration.

Table 1: Hardware Requirements

Component	Minimum Requirement	Recommended
OS	Windows 10/11 OR Google Colab Linux VM	Windows 11
RAM	12–16 GB	≥16 GB
CPU	Intel i7 9th Gen / Ryzen 5	i7 10th Gen / Ryzen 7
GPU	Google Colab T4 GPU	RTX 3050 / 3060
Storage	50 GB free	100 GB

Note:

- GPU acceleration is required for real-time deepfake detection.
- Minimum 12GB RAM required for simultaneous model loading

- SSD storage recommended for faster dataset I/O operations
- Stable internet connection required for Google Colab deployment
- Multiple CPU cores enable parallel preprocessing

3. Software Requirements

All development and execution were carried out in Python 3.10 using Google Colab environments, consistent with the tool usage seen in the samples.

Table 2: Python Libraries

Library	Version	Install Command
NumPy	1.26	Pip install NumPy
pandas	2.1	Pip install pandas
OpenCV-python	4.8	pip install OpenCV-python
matplotlib	3.8	pip install matplotlib
seaborn	0.12	Pip install seaborn
librosa	0.10	Pip install librosa
TensorFlow	2.14	Pip install TensorFlow
keras	2.14	pip install keras
speechbrain	0.5.15	Pip install speechbrain
scikit-learn	1.3	Pip install scikit-learn
dlib	19.24	Pip install dlib
streamlit	latest	Pip install streamlit
flask	latest	Pip install flask

These libraries support deepfake detection, biometric fusion, voice analysis, and dashboard creation.

4. Mount Google Drive

Purpose: Setting up Google Drive access is vital for loading cloud datasets and automatically saving all processed data, models, and results.

```
# Step 1: Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

5. Project Configuration

This section explains the dataset structure, folder hierarchy, reproducibility steps, and code configuration, mirroring the clarity of the sample manuals.

```
Project Root/
│
├── config/
│   └── config.py
│
├── code/
│   ├── face_recognition.py
│   ├── deepfake_detector.py
│   ├── voice_authentication.py
│   ├── behavioural_biometrics.py
│   ├── liveness_detection.py
│   ├── multimodal_fusion.py
│   ├── vulnerability_dashboard.py
│   └── main_pipeline.py
│
├── data/
│   ├── face/
│   │   ├── real/
│   │   └── fake/
│   ├── voice/
│   │   ├── real/
│   │   └── fake/
│   ├── behaviour/
│   ├── liveness/
│   └── synthetic_attacks/
│
├── models/
│   ├── deepfake_efficientnet.h5
│   ├── face_encoding.pkl
│   ├── voice_siamese_model.pt
│   ├── behavioural_svm.pkl
│   └── liveness_model.h5
│
└── output/
    ├── predictions.csv
    ├── logs/
    └── vulnerability_scores.json
```

5.1 Dataset Preprocessing

- **Face datasets**: FaceForensics++, DFDC(www.kaggle.com)
- **Voice datasets**: Synthetic deep-fake samples (Descript, pretrained TTS)
- **Behavioral data**: Keystroke/mouse dynamics
- **Synthetic attacks**: Generated via FaceSwap/DeepFaceLab
-

Datasets must be downloaded manually (20–50GB).

5.2 Configuration Variables

```
# Paths
FACE_DATA_PATH = "./data/face/"
VOICE_DATA_PATH = "./data/voice/"
BEHAV_DATA_PATH = "./data/behaviour/"
MODEL_PATH = "./models/"
OUTPUT_PATH = "./output/"

# Core toggles
RUN_FACE_RECOGNITION = True
RUN_VOICE_RECOGNITION = True
RUN_BEHAVIORAL_BIOMETRICS = True
RUN_DEEPFAKE_DETECTION = True
GENERATE_VULNERABILITY_SCORE = True

# Preprocessing
IMAGE_SIZE = (128, 128)
SAMPLE_RATE = 16000  # for audio

# ML parameters
EPOCHS = 20
BATCH_SIZE = 32
THRESHOLD_LIVENESS = 0.65
THRESHOLD_ATTACK = 0.40
```

These toggles mimic the Boolean flow-control switches in the sample manuals' configuration sections.

5.3 Sequence to Run the Code

Run preprocessing scripts

Extract faces

Normalize and resize images

Extract MFCC features for voice

Capture typing and mouse behavioral biometrics

Train/fine-tune deep-fake detectors

XceptionNet

EfficientNetB0

Run multi-modal biometric pipeline (main_pipeline.py)

Discussion

Library Import

```
[39]
# Step 2: Import necessary libraries
import os
import numpy as np
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from sklearn.model_selection import train_test_split
```

This code brings in critical libraries of machine learning with images.

OS manipulates file names, NumPy manipulates arrays of numbers, and deep-learning programs, especially the preprocessing ones offered by TensorFlow, process images by loading them.

Scikit-learn has train test split that divides data into testing and training sets that are used in building and testing models respectively.

Path Define

```
[40]
# Step 3: Define paths
dataset_path = "/content/drive/MyDrive/Deepfake/Final"
categories = ["Real", "Fake"]
```

This code sets the path to the dataset stored in Google Drive and defines the two image classes: Real and Fake.
These categories will be used to label images during loading and preprocessing for the deepfake detection model.
Load Image and Labels

```
# Step 4: Load images and labels
image_size = (128, 128)  # Resize all images to 128x128
X = []
y = []

for label, category in enumerate(categories):
    folder_path = os.path.join(dataset_path, category)
    for img_name in os.listdir(folder_path):
        img_path = os.path.join(folder_path, img_name)
        try:
            img = load_img(img_path, target_size=image_size)
            img_array = img_to_array(img) / 255.0  # Normalize to 0-1
            X.append(img_array)
            y.append(label)  # 0 = Real, 1 = Fake
        except Exception as e:
            print(f"Could not process {img_path}: {e}")
```

This code loads all images from the **Real** and **Fake** folders, resizes them to **128×128**, and converts them to numeric arrays normalized between 0–1. Each image is stored in **X**, while its label (0 = Real, 1 = Fake) is stored in **Y**. Errors during loading are caught and printed.

Shape of Image

```
# Convert to numpy arrays
X = np.array(X)
y = np.array(y)

print(f"Images shape: {X.shape}")
print(f"Labels shape: {y.shape}")
```

```
Images shape: (206, 128, 128, 3)
Labels shape: (206,)
```

This code converts the loaded image list **X** and label list **Y** into NumPy arrays. The printed shapes show **206 images**, each sized **128×128×3**, and **206 labels**, confirming the dataset was loaded correctly and is ready for model training.

Training and Test Sample

```
# Step 5: Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print(f"Training samples: {X_train.shape[0]}")
print(f"Testing samples: {X_test.shape[0]}")
```

```
Training samples: 164
Testing samples: 42
```

This code splits the dataset into **training (80%)** and **testing (20%)** sets using train_test_split. Stratification ensures both classes (Real, Fake) remain balanced in each split. The result: **164 training images** and **42 testing images**, ready for model training and evaluation.

Authentication and Vulnerability Score

```
sample_face = X_images[0]
sample_voice = np.random.rand(40)
sample_behaviour = np.random.rand(2)

score = multimodal_verification(sample_face, sample_voice, sample_behaviour)
risk = vulnerability_score(sample_face)

print("Authentication Score:", score)
print("Vulnerability Risk Score:", risk)
```

[53]

```
Authentication Score: 0.8626433593941383
Vulnerability Risk Score: 0.857362212193644
```

This code tests a multimodal authentication function using a sample face image, randomly generated voice features, and behavioural data.
It calculates two outputs:
Authentication Score: confidence that the user is genuine.

Vulnerability Risk Score: how susceptible the face sample is to be spoofing or deepfake attacks.

Training

```
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
```

[58] Python

```
history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=10,
    batch_size=32
)
```

[59] Python

```
Epoch 1/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 8s 913ms/step - accuracy: 0.5858 - loss: 0.7330 - val_accuracy: 0.7143 - val_loss: 0.6892
Epoch 2/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 8s 1s/step - accuracy: 0.7001 - loss: 0.6616 - val_accuracy: 0.7143 - val_loss: 0.6588
Epoch 3/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 8s 937ms/step - accuracy: 0.6925 - loss: 0.6214 - val_accuracy: 0.7143 - val_loss: 0.6726
Epoch 4/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 10s 811ms/step - accuracy: 0.7021 - loss: 0.6061 - val_accuracy: 0.7381 - val_loss: 0.6181
Epoch 5/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.7524 - loss: 0.5758 - val_accuracy: 0.7381 - val_loss: 0.6009
Epoch 6/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 9s 800ms/step - accuracy: 0.7481 - loss: 0.5631 - val_accuracy: 0.7381 - val_loss: 0.6062
Epoch 7/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 8s 1s/step - accuracy: 0.7496 - loss: 0.5520 - val_accuracy: 0.7381 - val_loss: 0.6645
Epoch 8/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 8s 986ms/step - accuracy: 0.7944 - loss: 0.4971 - val_accuracy: 0.5476 - val_loss: 0.7444
Epoch 9/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 10s 953ms/step - accuracy: 0.7493 - loss: 0.5343 - val_accuracy: 0.6905 - val_loss: 0.7623
Epoch 10/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 6s 931ms/step - accuracy: 0.7912 - loss: 0.4789 - val_accuracy: 0.6190 - val_loss: 0.9221
```

This output shows your deepfake detection model training for 10 epochs using binary cross-entropy and the Adam optimizer.
Training accuracy improves from 0.58 → 0.79, but validation accuracy fluctuates around 0.71–0.74, later dropping to 0.62 with rising validation loss—indicating overfitting on the small dataset (206 images).

CNN Accuracy

```
loss, acc = model.evaluate(X_test, y_test)
print("Accuracy:", acc)
```

```
[60]
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 139ms/step - accuracy: 0.6314 - loss: 0.8625
Accuracy: 0.6190476417541504
```

This evaluation shows the model's performance on unseen test data.
It achieves **63% accuracy**, meaning it correctly classifies about two-thirds of images as Real or Fake.
The high-test loss (**0.86**) indicates the model struggles to generalize, mainly due to **small dataset size and overfitting**.

Training and Validation Accuracy

```
import matplotlib.pyplot as plt

plt.figure(figsize=(8,6))
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title("Training vs Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.show()
```

Training accuracy keeps improving, but validation accuracy becomes unstable and drops sharply after epoch 6. This means the model is learning the training data too well and cannot generalize to new, unseen data, likely because the dataset is too small.

Confusion Matrix

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

y_pred = (model.predict(X_test) > 0.5).astype("int32")

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Real","Fake"])
disp.plot(cmap="Blues")
plt.title("Confusion Matrix")
plt.show()
```

The confusion matrix indicates superior classification performance for the Fake class compared to the Real class. The model's low recall on Real images is a symptom of class imbalance and confirms model overfitting.

ROC Curve

```
from sklearn.metrics import roc_curve, auc

y_pred_prob = model.predict(X_test)

fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.3f}")
plt.plot([0,1], [0,1], linestyle="--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.grid(True)
plt.show()
```

This ROC curve assesses how well the model distinguishes real from fake images. With an AUC of 0.472 (below the 0.5 random baseline), the model performs worse than chance. The curve hugs the diagonal line, indicating poor discrimination likely caused by overfitting on a small dataset.

Fake vs Actual Prediction

```
import random

plt.figure(figsize=(10,10))
for i in range(9):
    idx = random.randint(0, len(X_test)-1)
    img = X_test[idx]
    label = y_test[idx]
    pred = "Fake" if model.predict(img.reshape(1,128,128,3))[0][0] > 0.5 else "Real"

    plt.subplot(3,3,i+1)
    plt.imshow(img)
    plt.title(f"Actual: {'Fake' if label==1 else 'Real'} | Pred: {pred}")
    plt.axis('off')

plt.show()
```

```
loss, acc = model.evaluate(X_test, y_test)
print("Accuracy:", acc)
```

```
[60]
...  2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 139ms/step - accuracy: 0.6314 - loss: 0.8625
Accuracy: 0.6190476417541504
```

---

6. Artefact Details

The submitter is limited in size (datasets of the order of 40GB, generated images of the order of 3GB), which is why, similarly to the sample manuals, large datasets are not provided.

Artefacts included:

/code/ – all Python source files

/models/ – pretrained models under 200MB

/output/ – sample risk reports, logs, and prediction files

Artefacts NOT included:

DFDC dataset

FaceForensics++ dataset

Synthetic deepfake videos

Large image datasets created during preprocessing:

7. Configuration Variables

Variable	Default	Description
RUN_FACE_RECOGNITION	True	Enables face ID + encoding
RUN_VOICE_RECOGNITION	True	Enables voice biometric Siamese model
RUN_BEHAVIORAL	True	Runs keystroke + mouse biometrics
RUN_DEEPFAKE_DETECTION	True	Enables CNN deepfake model
RUN_LIVENESS_DETECTION	True	Detects eye-blink, head movement
GENERATE_VULNERABILITY_SCORE	True	Outputs risk metrics
RUN_DASHBOARD	True	Launches Streamlit dashboard

7.1 Core Toggles

Variable	Default	Description
RUN_FACE_RECOGNITION	True	Enables face ID + encoding
RUN_VOICE_RECOGNITION	True	Enables voice biometric Siamese model
RUN_BEHAVIORAL	True	Runs keystroke + mouse biometrics
RUN_DEEPFAKE_DETECTION	True	Enables CNN deepfake model
RUN_LIVENESS_DETECTION	True	Detects eye-blink, head movement
GENERATE_VULNERABILITY_SCORE	True	Outputs risk metrics
RUN_DASHBOARD	True	Launches Streamlit dashboard

7.2 Biometric Processing Parameters

Parameter	Value
IMAGE_SIZE	(128, 128)
AUDIO_SAMPLE_RATE	16000
MFCC_FEATURES	40
BEHAV_WINDOW_SIZE	5 seconds
LIVENESS_THRESHOLD	0.65
DEEPFAKE_THRESHOLD	0.40

7.3 Training Parameters

Parameter	Default
EPOCHS	20

BATCH_SIZE	32
OPTIMIZER	Adam
LOSS	Binary Crossentropy

8. Modalities Setup

8.1 Facial Recognition Configuration

Dependencies: OpenCV, Dlib, face_recognition

Steps:

Place all face images in:

```
/data/face/real/
/data/face/fake/
```

Run face alignment using face_recognition.face_landmarks

Generate facial embeddings (face_encoding.pkl)

Compare new input embedding against the stored embedding using cosine similarity.

8.2 Voice Authentication Configuration

Dependencies: librosa, speechbrain

Steps:

Store audio files in:

```
/data/voice/real/
/data/voice/fake/
```

Extract MFCC features using:

```
librosa.feature.mfcc(n_mfcc=40)
```

Use a pretrained **Siamese Network** for similar scoring.

Output:

Voice Authentication Score

Spoof Detection Score

8.3 Behavioural Biometrics Configuration

Dependencies: pynput, scikit-learn

Modalities:

Keystroke dynamics

Mouse movement patterns

Steps:

Install:

```
pip install pynput
```

Capture features:

Key press duration

Flight time between keys

Mouse acceleration

Direction vectors

Train SVM classifier → behavioural_svm.pkl

8.4 Liveness Detection Configuration

Methods used:

Eye-blink detection

Head-movement

Texture-based spoof detection

Steps:

Use OpenCV Haar cascades

Track EAR (Eye Aspect Ratio)

Threshold EAR < 0.20 → blink confirmed

Generate liveness score

---

9. Deepfake Detection Model (Already Implemented)

Now integrated into the multimodal pipeline.

---

10. Vulnerability Dashboard Setup

Dependencies:

```
pip install streamlit
```

Run Dashboard:

```
streamlit run vulnerability_dashboard.py
```

Dashboard Displays:

TAR (True Acceptance Rate)

FAR (False Acceptance Rate)

EER (Equal Error Rate)

ASR (Attack Success Rate)

Modality-wise risk score

Fusion risk score

Trend charts over sessions

---

11. Running the Full Multi-Modal Pipeline

Run:

```
python main_pipeline.py
```

Pipeline steps:

Face recognition

Voice authentication

Behavioural biometrics

Liveness detection

Deepfake detection

Multi-modal fusion

Vulnerability scoring

Dashboard update

12.Troubleshooting

Issue	**Fix**
Dlib installation fails	Install C++ Build Tools
TensorFlow no GPU	Enable GPU in Colab Runtime
Path not found	Update BASE_PATH in config.py
MFCC extraction crash	Ensure files are 16kHz WAV
Streamlit error	Run from local terminal, not Colab

12. Artefacts Included vs Excluded

Included

All Python scripts

Pretrained small models

Logs, prediction files

Not Included

DFDC dataset

FaceForensics++

Deepfake video files

---

14. Vulnerability Assessment Dashboard

This section provides the full explanation, configuration steps, code structure, and execution procedure for the Vulnerability Assessment Dashboard as a key deliverable for the thesis.

The dashboard provides real-time visualization of biometric system risks, deepfake susceptibility, accurate metrics, and modality-wise performance. It is implemented using Streamlit and reads outputs generated by the multi-modal pipeline.

14.1 Purpose of the Dashboard

The Vulnerability Dashboard allows financial institutions, developers, and auditors to:

Measure how secure the biometric system **is** under synthetic attack scenarios.

Visualize risk scores generated by the deepfake detector, liveness model, behavioural classifier, and voice model.

Track metrics over time (comparisons between real, fake, and spoof attempts).

Identify weak biometric modalities (face, voice, behaviour, liveness).

Assess the Attack Success Rate (ASR) and generate alerts for high-threat activity.

Assist in operational decisions regarding KYC onboarding and fraud detection.

14.2 Installation & Requirements

Install Streamlit

```
pip install streamlit
```

Install supporting libraries

```
pip install matplotlib seaborn scikit-learn pandas NumPy
```

13.4 Configuration Variables for the Dashboard

```
DASHBOARD_DATA_PATH = OUTPUT_PATH + "metrics/"
VULNERABILITY_FILE = OUTPUT_PATH + "vulnerability_scores.json"
PREDICTIONS_FILE = OUTPUT_PATH + "predictions.csv"
DASHBOARD_PORT = 8501
```

These variables ensure the dashboard works on:

Local Windows systems

Linux

Google Colab (via public tunneling)

14.3 Core Functions in vulnerability_dashboard.py

The dashboard uses five core modules:

1. load_metrics()

Loads JSON files for each modality:

face_metrics.json

voice_metrics.json

behaviour_metrics.json

liveness_metrics.json

deepfake_metrics.json

2. load_vulnerability_score()

Loads combined risk score:

Authentication Score

Vulnerability Score

ASR

Modality Confidence Levels

3. render_modality_cards()

Displays:

Accuracy

Precision

Recall

Spoof Detection Rate

for each modality.

4. render_risk_graphs()

Shows:

Risk over time

Histogram of deepfake scores

ROC curves (if available)

5. generate_risk_alert()

If Vulnerability Score > 0.60:

Display **HIGH RISK WARNING**

Highlight the weakest modality

14.4 Running the Dashboard

From the project root:

```
streamlit run code/vulnerability_dashboard.py
```

This launches the dashboard on:

Local:

http://localhost:8501

Colab:

You will receive a **public URL via ngrok** automatically (Streamlit provides).

14.5 Expected Outputs of the Dashboard

Once launched, the dashboard displays the following:

A. System Overview Panel

System Status: Operational / Under Attack

Current Authentication Score

Current Vulnerability Score

Last Updated Timestamp

B. Attack Detection Metrics

Metric	Description
TAR	True Acceptance Rate
FAR	False Acceptance Rate
ASR	Attack Success Rate
EER	Equal Error Rate

C. Modality-Wise Scores

Each modality displays:

Accuracy

Spoof Resilience Score

Confidence Level

Risk Level (Low / Medium / High)

D. Risk Heatmap

Shows which biometric components are most vulnerable.

Example:

Face	Voice	Behaviour	Liveness
High	Medium	Low	Medium

E. Deepfake Attack Insights

Real vs Fake score distribution

Bar chart of misclassifications

Sensitivity to synthetic perturbation (blur, colour distortion, GAN noise)

F. Trend Charts Over Sessions

Plots how risk changes across multiple verification attempts.

14.6 How the Dashboard Integrates with the Pipeline

After every authentication attempt, the main pipeline writes:

```
vulnerability_scores.json
```

Example file:

```
{
  "authentication_score": 0.84,
  "vulnerability_score": 0.61,
  "face_risk": 0.42,
  "voice_risk": 0.39,
  "behaviour_risk": 0.18,
  "liveness_risk": 0.33,
  "deepfake_probability": 0.52,
  "timestamp": "2025-01-23 14:32:10"
}
```

The dashboard reads this in real time to update:

Progress bars

Gauges

Heatmaps

Alerts

14.7 Reproducing the Dashboard

Step 1: Run the multi-modal authentication pipeline

```
python main_pipeline.py
```

Step 2: Ensure output files exist in /output/metrics/

Face, voice, behaviour, liveness, and deepfake models each generate JSON metric files.

Step 3: Start the Streamlit Dashboard

```
streamlit run code/vulnerability_dashboard.py
```

Step 4: View real-time vulnerability assessment results

Use the local browser or Streamlit's public URL.

14.8 Troubleshooting the Dashboard

Issue	Cause	Fix
"FileNotFoundError: metrics not found"	Missing evaluation step	Run main_pipeline.py first
Dashboard blank	JSON not properly formatted	Delete corrupted file → rerun pipeline
Streamlit not launching	Firewall or port conflict	Change port in config.py
Plots not appearing	Missing matplotlib	pip install matplotlib

References

Chollet, F. (2015). Keras: The Python Deep Learning library.

www.kaggle.com

Sumalatha, U., Prakasha, K.K., Prabhu, S. and Nayak, V.C., 2024. A comprehensive review of unimodal and multimodal fingerprint biometric authentication systems: Fusion, attacks, and template protection. *IEEE Access*, *12*, pp.64300-64334.