Project Title:

End-to-End Image Forgery Detection using Deep Learning & MLOps

Name and Email:

Sharan Muthu Krishna K – sharanmuthukrishna@gmail.com

GitHub:

https://github.com/Sharan-Muthu-Krishna/forgery-detection-end-to-end-pipeline.git

1. Introduction

Image forgery has become a serious problem with the rise of digital editing tools and AI-generated images. Forged images are widely used in misinformation, fake identities and document fraud.

This project presents a **production-ready AI system** that automatically detects whether an image is **Original** or **Forged** using a deep learning model combined with a full **MLOps pipeline**.

Unlike academic projects, this system is designed to operate like a real industry deployment with:

- Automated training
- Model tracking
- Performance-based deployment
- API serving
- User interface

2. Project Objective

The goals of this project are:

- Detect forged images using deep learning and Error Level Analysis (ELA)
- Build a full ML pipeline using ZenML
- Track experiments and models using MLflow
- Automatically deploy only better models
- Serve the model through FastAPI
- Provide a user-friendly UI with Streamlit

3. System Overview

The system consists of three major layers:

1. **Data ingestion pipeline**
2. **Training & Evaluation Pipeline**
3. **Prediction Application**

Every time the pipeline runs:

- A new model is trained
- It is evaluated on test data
- It is compared with the current production model
- It is deployed only if it performs better

This prevents bad models from reaching users.

4. Machine Learning Approach

The project uses:

- Error Level Analysis (ELA) to expose compression artifacts and highlight manipulated regions in images
- A MobileNet-based Convolutional Neural Network (CNN) fine-tuned for forgery detection using a Kaggle dataset and a custom ELA-processed dataset created specifically for this project.
- Binary classification:
    - 0 → Forged
    - 1 → Original

ELA highlights manipulated regions which are invisible in raw images but detectable by deep learning.

5. MLOps Architecture

This project follows modern MLOps practices.

Why ZenML?

ZenML manages the entire ML pipeline:

- Data preparation
- Model training
- Evaluation
- Deployment logic

It allows reproducibility and automation.

Why MLflow?

MLflow tracks:

- Model versions
- Metrics
- Training runs
- Deployed models

This allows comparing models and maintaining history.

Why automated deployment?

The pipeline automatically checks:

“Is the new model better than the current one?”

If yes → Deploy
If no → Reject

6. Technology Stack

Category	Technology
Language	Python
Deep Learning	TensorFlow / Keras
Image Processing	PIL (Python Imaging Library)
Pipeline Orchestration	ZenML
Experiment Tracking	MLflow
API	FastAPI
UI	Streamlit
Model Format	Keras (.keras)

7. Model Evaluation & Deployment

The model is evaluated using:

- Accuracy
- F1-Score

Why F1-Score instead of Accuracy?
Forgery detection is a sensitive task.
False positives and false negatives both matters.
F1-Score balances precision and recall, making it more reliable.

The production model is stored as:

models/production_model.keras

This file is always the **best model**.

8. Web Application

Two applications are used:

FastAPI

- Hosts /predict endpoint
- Loads latest production model
- Applies ELA
- Returns prediction and confidence

Streamlit

- User uploads image
- Calls FastAPI
- Displays result visually

9. Results

The final trained and deployed model achieved the following performance:

Training Performance

Metric	Value
Accuracy	**99.53%**
Loss	**0.0285**

These results indicate that the model learned the underlying forgery patterns extremely well on the training dataset.

Test (Unseen Data) Performance

Metric	Value
Test Accuracy	**92.37%**
Test F1-Score	**92.81%**

The strong test accuracy and F1-score confirm that the model generalizes well to new images and does not overfit.
The high F1-score shows that both **forged** and **original** images are classified reliably.

This deployed model will be stored as:

models/production_model.keras

and is always kept as the best-performing version by the automated deployment system.

10. How to Set Up and Run the Project

This project is designed to run locally with a complete MLOps pipeline

10.1 Requirements

- Install everything:

```
pip install -r requirements.txt
```

10.2 ZenML Setup

- Initialize ZenML inside the project folder:

```
zenml init
```

- Create MLflow tracking and model deployer:

zenml integration install mlflow

zenml experiment-tracker register mlflow_tracker --flavor=mlflow

zenml model-deployer register mlflow_deployer --flavor=mlflow

- Create and activate the ZenML stack:

zenml stack register local-mlflow-stack \

-a default \

-o default \

-e mlflow_tracker \

-d mlflow_deployer \

--set

10.3 Start MLflow UI

- mlflow ui
- Then open in browser:

http://localhost:5000

10.4 Run Data Ingestion & ELA Pipeline

- python run_pipeline.py
- This:
  - Cleans images
  - Generates ELA images
  - Prepares training data

10.5 Train, Evaluate & Deploy

- python run_train_eval_deploy.py
- This:
  - Trains a new model
  - Evaluates on test data
  - Compares with current production model
  - Deploys only if better

10.6 Start Prediction API

- cd serving/api
- uvicorn main:app --reload
- API will run at:

  http://127.0.0.1:8000
- Swagger UI:

  http://127.0.0.1:8000/docs

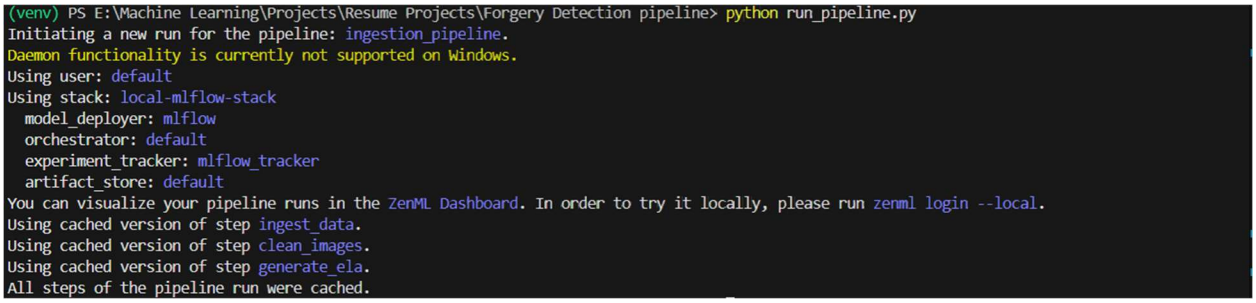
10.7 Start Web Interface

- Open a new terminal:
- cd serving/ui
- streamlit run app.py
- This opens a browser where you can upload images and get predictions.

11. Screenshots

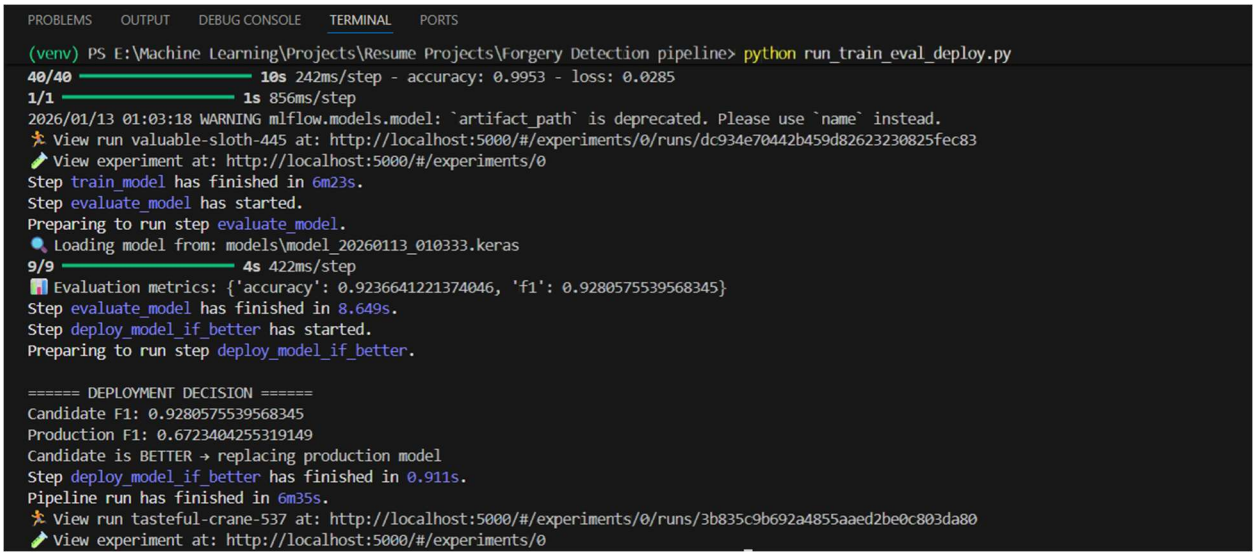
11.1 VS Code – Ingestion Pipeline Execution

This screenshot shows the successful execution of the data ingestion pipeline in VS Code.
It confirms that the raw dataset was cleaned, validated, and converted into Error Level Analysis (ELA) images, preparing the data for deep learning model training.



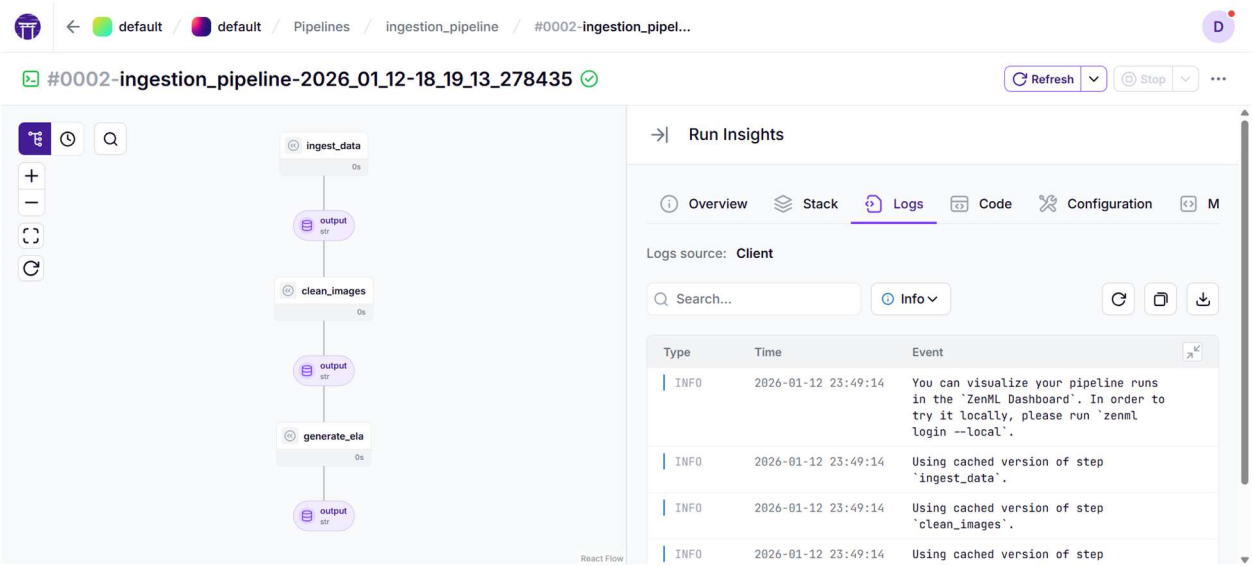
11.2 VS Code – Train, Evaluate & Deploy Pipeline

This screenshot displays the execution of the **automated training, evaluation, and deployment pipeline**.
It verifies that a new model was trained, evaluated on test data, compared with the current production model, and deployed only if it performed better.



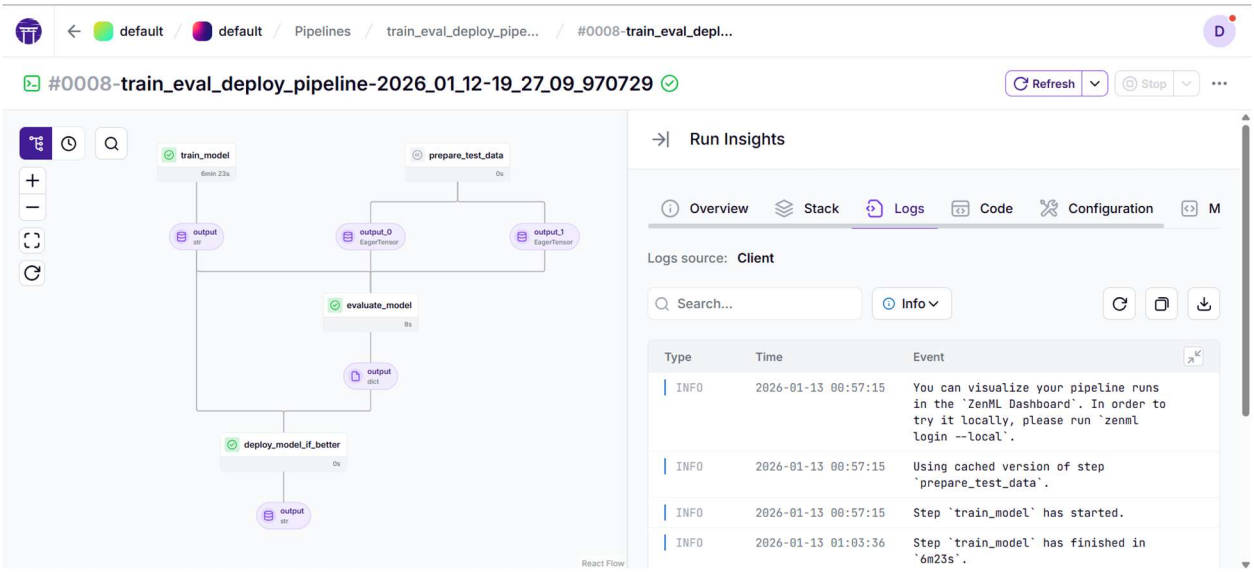
11.3 ZenML Dashboard – Ingestion Pipeline

This screenshot shows the **ZenML pipeline graph** for data ingestion.
Each step (image cleaning, preprocessing, and ELA generation) is tracked and versioned.



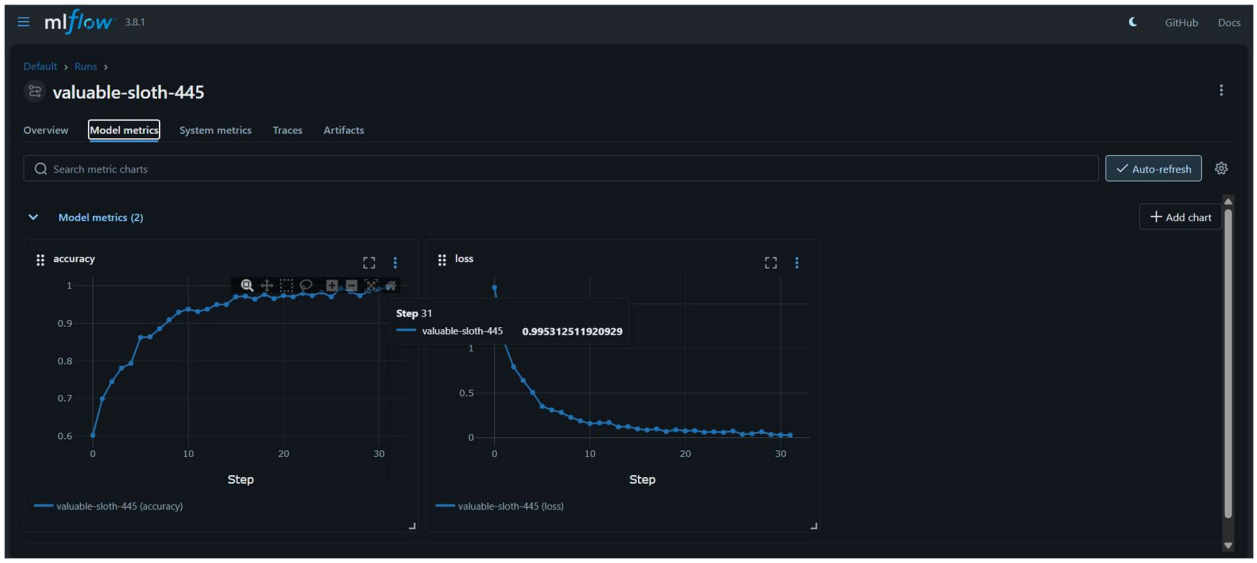
11.4 ZenML Dashboard – Train, Evaluate & Deploy Pipeline

This view presents the **end-to-end MLOps pipeline** including training, testing, and deployment decision logic.
It demonstrates how ZenML orchestrates multiple steps into a single automated workflow.



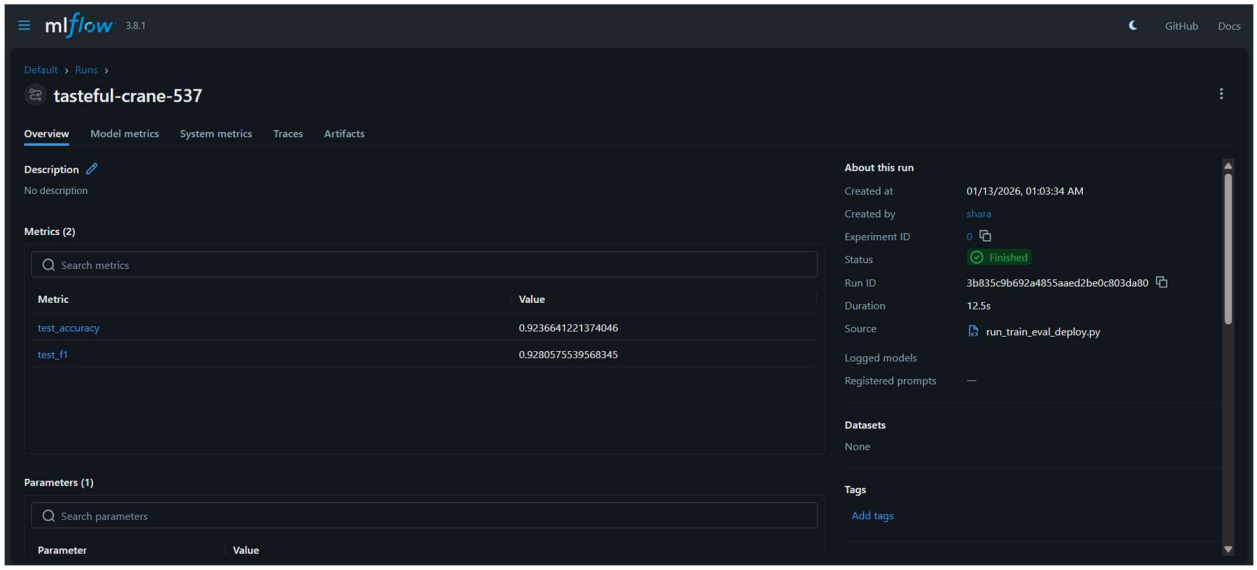
11.5 MLflow – Training Metrics

This screenshot shows **training metrics logged in MLflow**, including loss and accuracy.
It allows comparison between different training runs and helps track model improvement over time.



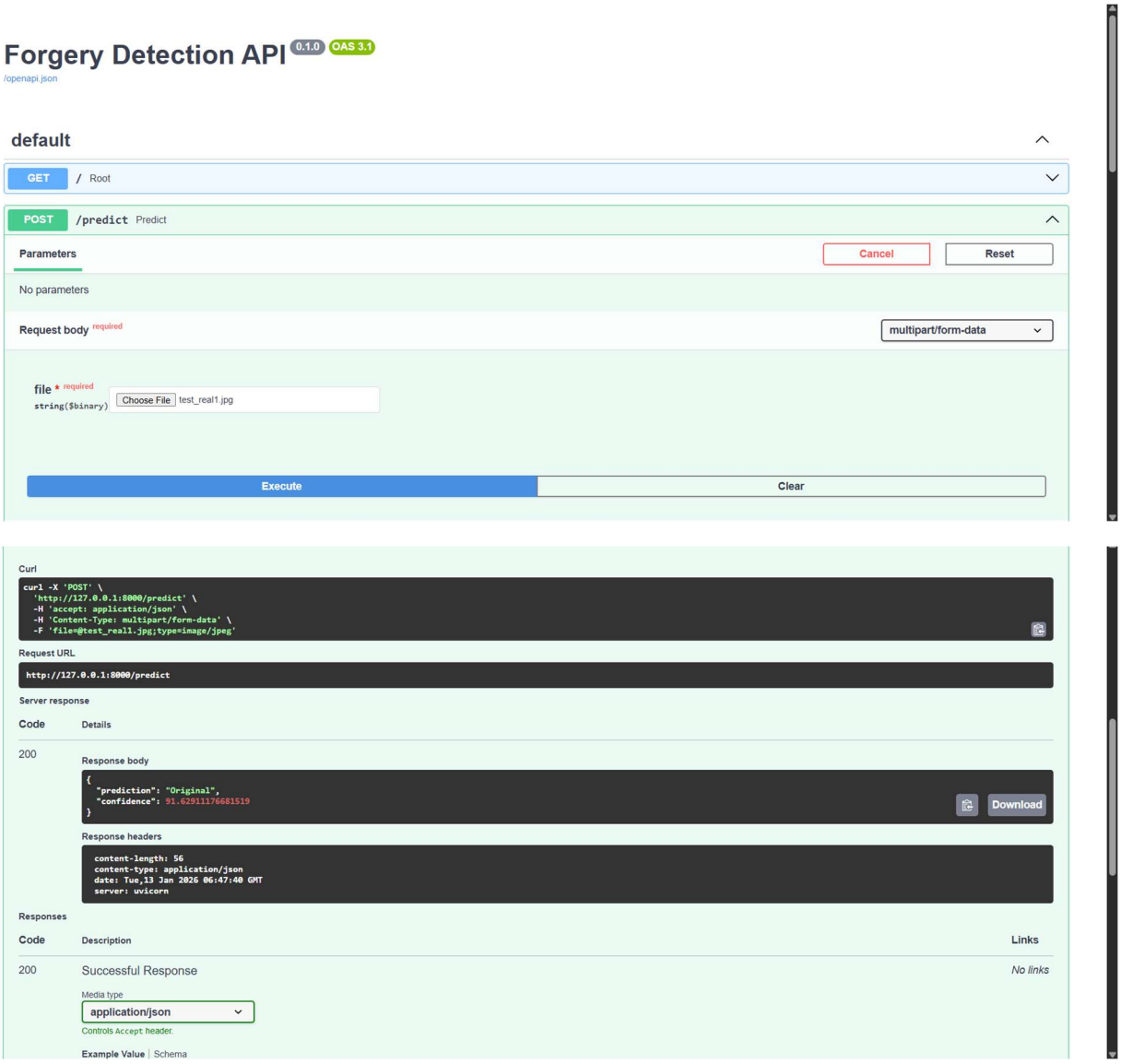
11.6 MLflow – Testing Metrics

This screenshot displays the **evaluation metrics** such as **test accuracy and F1-score** for the trained model.
These metrics are used by the deployment logic to decide whether the model should be promoted to production.



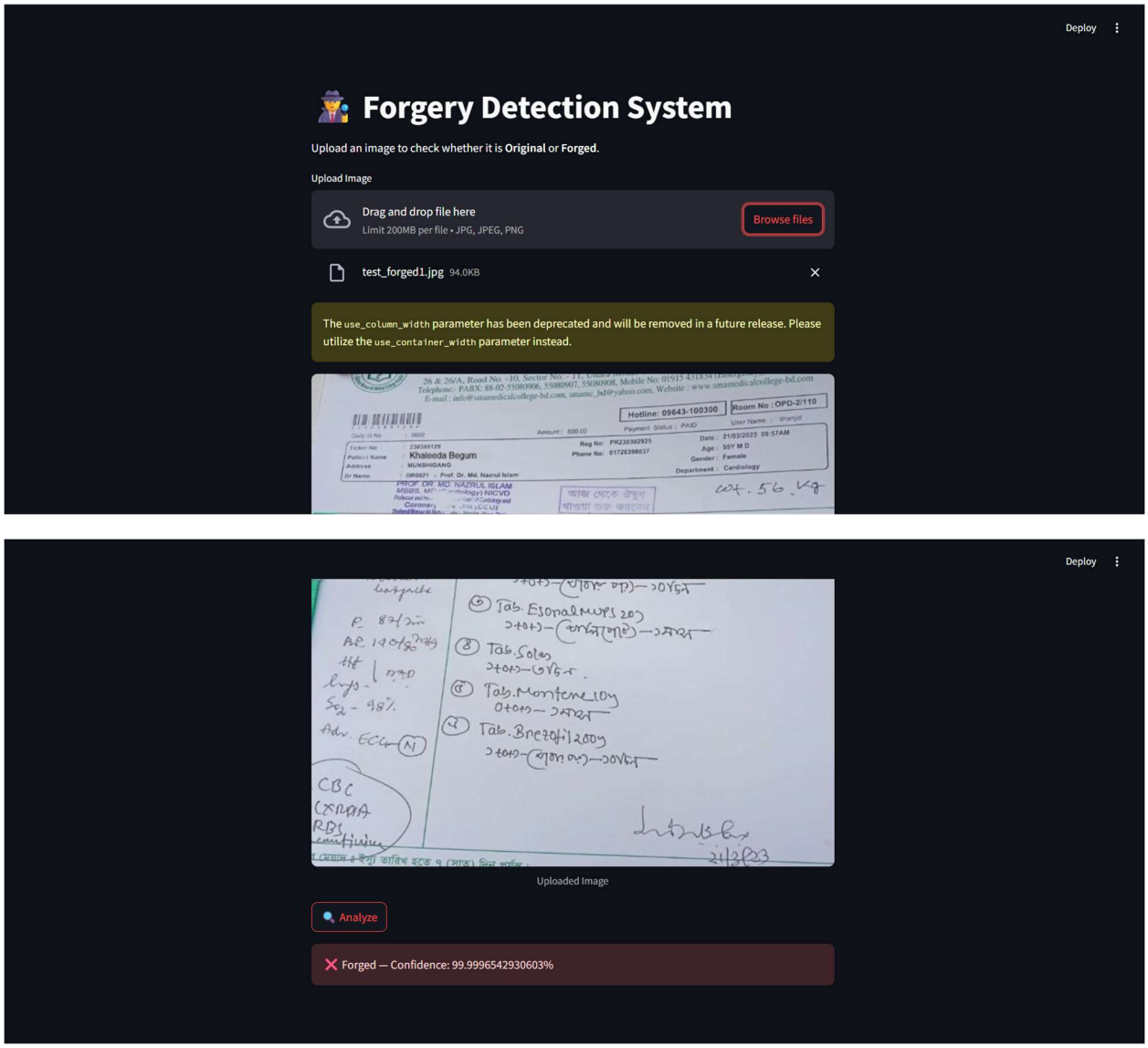
11.7 FastAPI Swagger UI

This screenshot shows the **Swagger interface** of the FastAPI prediction service. It allows users and developers to test the /predict endpoint by uploading an image and receiving forgery predictions in real time.



11.8 Streamlit Web Interface – Forged Image Detection

This screenshot shows the **Streamlit user interface** where a user uploads an image.
The system processes the image using the deployed model and correctly identifies it as **Forged**, displaying the prediction along with confidence.



12. Conclusion

This project successfully demonstrates a complete end-to-end **AI-powered forgery detection system** built using modern MLOps practices. By combining **Error Level Analysis (ELA)** with a deep learning model based on **MobileNetV2**, the system is able to detect image manipulations with high accuracy and reliability.

Beyond just model training, this project implements a full **production-grade machine learning lifecycle** using **ZenML and MLflow**. The pipeline automates data preprocessing, model training, evaluation, versioning, and deployment. A key feature of this system is its **performance-based deployment strategy**, where a newly trained model replaces the production model only if it achieves better evaluation metrics, ensuring continuous improvement without manual intervention.

The integration of a **FastAPI backend** and a **Streamlit web interface** allows real-time image analysis, making the system practical for real-world use cases such as document verification, digital forensics, and media authenticity checking.

Overall, this project showcases not only strong deep learning and computer vision skills, but also the ability to design and deploy **scalable, maintainable, and automated machine learning systems**.