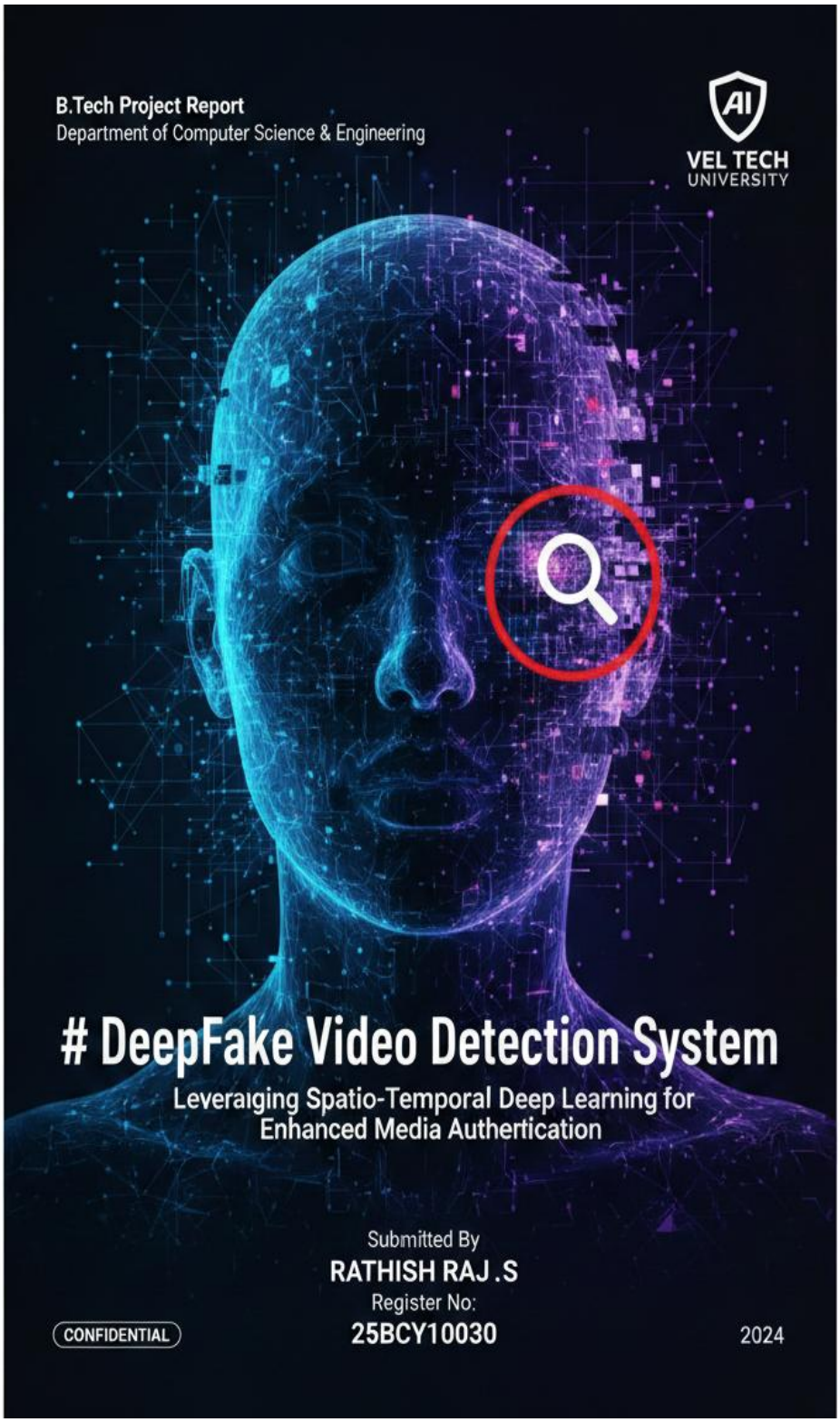
B.Tech Project Report
Department of Computer Science & Engineering

VEL TECH
UNIVERSITY



DeepFake Video Detection System

Leveraiging Spatio-Temporal Deep Learning for Enhanced Media Authertication

Submitted By
RATHISH RAJ .S
Register No:
25BCY10030

CONFIDENTIAL

2024

Introduction

The proliferation of highly realistic synthetic video content, commonly known as **DeepFakes**, presents one of the most critical challenges to digital trust and information security today. These advanced manipulations, often created using generative adversarial networks (GANs) and autoencoders, threaten public discourse by enabling the rapid spread of misinformation, the fabrication of false events, and sophisticated identity fraud. The ability to create videos where one person's face or voice is convincingly superimposed onto another's body has made effective media authentication a necessity.

To combat this escalating threat, the **DeepFake Video Detection System** is developed as an automated, high-confidence solution for verifying video authenticity. This system leverages state-of-the-art **Computer Vision** and **Deep Learning** techniques, primarily employing a **spatio-temporal model**. This specialized architecture is designed to go beyond simple visual inspection, identifying subtle, non-human artifacts and inherent temporal inconsistencies—such as unnatural motion or flickering—that are characteristic fingerprints of generative video manipulation.

This project focuses on applying core Machine Learning concepts—specifically **classification and prediction**—to solve this critical, real-world problem, thereby enhancing digital security and restoring confidence in visual media.

Problem Statemen

The rapid advancement and widespread accessibility of generative AI models have led to the proliferation of highly realistic synthetic video content, commonly referred to as **DeepFakes**. These manipulated videos, created through sophisticated techniques like Generative Adversarial Networks (GANs) and autoencoders, pose a critical and escalating threat to digital trust, security, and the integrity of information.

The core problem is twofold:

The Threat to Trust and Security: DeepFakes enable the large-scale dissemination of convincing misinformation, facilitate sophisticated identity fraud (e.g., bypassing biometric security or impersonating individuals for financial gain), and are used for malicious content creation, thereby eroding public confidence in the authenticity of video evidence.

The Ineffectiveness of Current Methods: Traditional video authentication and manual review processes are incapable of reliably distinguishing these subtle, algorithmically-generated manipulations from authentic footage. The artifacts are too nuanced and temporal inconsistencies too fleeting for the human eye or rudimentary software to consistently detect.

Therefore, the objective problem requiring a technical solution is:

To develop an automated, high-confidence DeepFake Video Detection System that can reliably and accurately classify video footage as either authentic or fake. This system must leverage advanced Computer Vision and Deep Learning techniques, specifically a **spatio-temporal model**, to detect the subtle, non-human artifacts and temporal inconsistencies inherent in generative manipulations, thereby restoring a measure of accountability and trust in digital media.

Functional Requirements

The functional requirements for the **DeepFake Video Detection System** are organized into three primary modules for a robust classification and reporting workflow:

- **Module 1: Video Ingestion and Preprocessing**
    - **FR1 (Video Ingestion):** The system must accept standard video file formats (e.g., MP4, AVI) as input.
    - **FR2 (Face Localization and Normalization):** The system must utilize computer vision algorithms (like MTCNN/Dlib) to detect, track, crop, and normalize human faces in every frame to a consistent size (e.g., 256x256).
- **Module 2: Spatio-temporal Model Inference Engine**
    - **FR3 (Spatio-temporal Feature Extraction):** The system must use a CNN component to extract spatial features (texture, artifacts) from each normalized face frame.
    - **FR4 (Temporal Analysis and Classification):** The system must use a recurrent/transformer component (e.g., LSTM/Vision Transformer) to analyze the sequence of spatial features for unnatural temporal discontinuities, producing a "Fake Probability" for each frame.
- **Module 3: Results Reporting and Visualization**
    - **FR5 (Verdict Generation):** The system must calculate an aggregate probability score for the entire video and issue a final **REAL** or **FAKE** binary classification.
    - **FR6 (Time-series Reporting):** The system must generate a visualization (e.g., a Matplotlib plot) showing the probability of manipulation over the video's timeline, providing precise identification of manipulated segments.
    - **FR7 (System Logging):** The system must log the analysis details (filename, timestamp, final verdict, aggregate score) for auditing purposes.

.

Non-functional Requirements

The non-functional requirements (NFRs) for the DeepFake Video Detection System focus on the quality, performance, security, and maintainability of the technical solution.

- **Reliability (Accuracy):**
    - **The core classification model must achieve an Area Under the ROC Curve (AUC) score of at least 0.95 on a hold-out test set to ensure a low rate of false positives and negatives.**
- **Performance (Speed):**
    - **The system must process videos at a minimum speed of 10 frames per second (FPS) to ensure timely analysis and a quick user experience for media verification tasks.**
- **Security (Data Handling):**
    - **All uploaded video files must be temporarily stored using AES-256 encryption and automatically purged from the server 60 seconds after the analysis is complete to protect user data and privacy.**
- **Maintainability (Code Quality):**
    - **All code must adhere to PEP 8 standards, and the three major functional modules must be implemented as separate, distinct Python classes/files with clear APIs to simplify future model updates, debugging, and code maintenance.**

System Architecture

The System Architecture for the **DeepFake Video Detection System** is based on a **Three-Layer Architecture** to ensure modularity, scalability, and efficient processing of the computationally intensive Deep Learning components.

System Architecture (Three-Layer Model)

- **1. Presentation/Client Layer (Input/Output)**
    - **Role:** Handles all user interaction, including file submission and result display.
    - **Components:** A simple GUI (e.g., built with Tkinter or a web interface using Streamlit/Flask) for video upload.
    - **Function:** Takes the video file path from the user and presents the final classification verdict and graphical reports.
- **2. Application Logic Layer (Controller & Orchestration)**
    - **Role:** Manages the entire analytical workflow and integrates the specialized modules.
    - **Functional Modules: Module 1 (Preprocessing)** and **Module 3 (Reporting)**.
    - **Processes:**
        - Coordinates the execution flow between all modules.
        - Manages video file I/O and initiates face detection/cropping.
        - Generates the final **REAL/FAKE verdict** based on the model scores.
        - Creates the **time-series probability plot** and manages the **system audit log**.
- **3. Core ML/Data Layer (Inference Engine)**
    - **Role:** Executes the primary classification task, requiring significant computational power.

- **Functional Module: Module 2 (Inference Engine)**.
- **Components:** The trained Spatio-temporal Deep Learning model (e.g., **Convolutional Vision Transformer - CVT**).
- **Resource Requirement:** Typically requires dedicated accelerators (GPU/TPU) for fast inference.
- **Function:** Receives normalized face tensors and produces the frame-wise probability scores for manipulation.

Design Diagrams

The Design Diagrams visually translate the conceptual architecture and requirements into actionable blueprints for the **DeepFake Video Detection System**

1. Use Case Diagram

The Use Case Diagram illustrates the system's primary functionality from the perspective of the external **Analyst** (or user), showing how they interact with the three major functional modules.

- **Actor:** Analyst (User)
- **Primary Use Cases:**
  - **Submit Video:** Initiate the analysis by uploading a video file.
  - **Get Verdict:** Receive the final REAL or FAKE classification.
  - **View Report:** Access the time-series probability plot.
  - **Query Logs:** Review the system's audit log for past analysis

2. Workflow Diagram (Process Flow)

The Workflow Diagram shows the high-level sequence of data processing and control flow across the three required functional modules.

1. **Input:** Analyst submits the video.
2. **Module 1: Preprocessing:** The video is broken down into frames, and faces are detected and normalized.
3. **Module 2: Inference:** The normalized frames are fed into the Spatio-temporal Model, which performs **Classification** to generate frame-wise scores.
4. **Decision Point:** The aggregate score is checked against the threshold (e.g., 0.5).
5. **Module 3: Reporting:** The verdict is determined, and the time-series plot is generated and saved.
6. **Output:** The final result is displayed to the Analyst.

3. Sequence Diagram (Video Submission)

The Sequence Diagram details the interaction and messaging between the key system components when a video submission is processed.

- The **Analyst** sends the video file to the **GUI (Detector)**.
- The **GUI** initiates the process by calling the preprocess_video() method in the **Preprocessor Module**.
- After receiving the processed frames, the **GUI** calls run_inference() in the **Inference Engine**.
- The **Inference Engine** returns the classification scores to the **GUI**.
- The **GUI** then calls generate_report() in the **Reporting Module** to finalize the analysis and display the verdict to the **Analyst**

4.Class/Component Diagram

The Class Diagram illustrates the modular, object-oriented structure of the code, showing the main controller class and its reliance on the three specialized modules. This emphasizes the **Maintainability** NFR and the clean separation of concerns.

- **DeepFakeDetector** is the central controller, holding the state of the loaded model.
- It utilizes the **PreprocessorModule**, **InferenceEngine**, and **ReportingModule** to execute its workflow methods.

Design Decisions & Rationale

- **Spatio-temporal Architecture (CNN + LSTM/CVT)**
    - **Decision:** Use a hybrid model that combines a **Convolutional Neural Network (CNN)** for spatial feature extraction with a **Long Short-Term Memory (LSTM)** or **Vision Transformer (CVT)** for temporal analysis.
    - **Rationale:** DeepFakes contain two distinct types of artifacts: **spatial** (textures, blending at face boundaries) and **temporal** (unnatural flicker, inconsistent head movements, or lack of blinking). This combined architecture is necessary to detect both types simultaneously, leading to a robust classification.
    - **Requirement Adherence:** Directly addresses **FR3** (Spatial Extraction), **FR4** (Temporal Classification), and supports **NFR1** (Reliability ≥ 0.95 AUC).
- **Modular Codebase (Three Distinct Modules)**
    - **Decision:** Implement the system logic across three distinct, decoupled modules: PreprocessorModule, InferenceEngine, and ReportingModule.
    - **Rationale:** This promotes **modularity and maintainability**. By separating the workflow steps, the core ML model (Module 2) can be updated or replaced without requiring changes to the data preparation (Module 1) or output generation (Module 3).
    - **Requirement Adherence:** Meets the core project requirement for **Three major functional modules** and supports **NFR4** (Maintainability).
- **Face Normalization (Fixed Size Input)**
    - **Decision:** Use computer vision libraries (Dlib/MTCNN) to precisely detect, crop, and normalize only the face region of interest to a fixed size (e.g., 256×256 pixels).
    - **Rationale:** This removes irrelevant background noise that could confuse the model and significantly reduces the total data size

processed by the ML model. This is critical for achieving required processing speeds.

    - **Requirement Adherence:** Directly addresses **FR2** (Face Localization and Normalization) and supports **NFR2** (Performance ≥ 10 FPS).

- **AUC as Primary Evaluation Metric**
    - **Decision:** Use the **Area Under the ROC Curve (AUC)** instead of simple accuracy to evaluate model performance.
    - **Rationale:** DeepFake datasets are often **imbalanced** (more real videos than fake, or vice-versa). AUC is a superior metric because it measures the model's ability to distinguish between classes across all possible thresholds, providing a more reliable measure of the system's true predictive power.
    - **Requirement Adherence:** Directly satisfies the core metric defined in **NFR1** .

Implementation Details

1. Technology Stack and Libraries

The project relies on a focused set of Python technologies optimized for Computer Vision and Deep Learning performance:

- **Programming Language: Python 3.x**
- **Deep Learning Framework: TensorFlow** or **PyTorch** (Used to define and execute the core model).
- **Computer Vision Libraries:**
    - **OpenCV (cv2)**: Used for all **video file handling** (reading the video stream and extracting frames).
    - **Dlib / MTCNN**: Employed for robust **face detection, localization, and tracking** across frames.
- **Data Handling: NumPy** for high-performance numerical operations and efficient manipulation of image tensors.
- **Reporting: Matplotlib** for generating the visual **time-series probability plot** (FR6).

2. Modular Code Structure

The project is broken down into distinct files, strictly aligning with the three major functional modules to enforce **NFR4 (Maintainability)**:

- **deepfake_detector.py (Main Controller):**
    - Serves as the main entry point and orchestrator.
    - Initializes the system, loads the pre-trained model weights, and coordinates calls to the three modules.
- **preprocessing.py (Module 1):**
    - Contains the PreprocessorModule class.
    - Responsible for the preprocess_video(video_path) method, which returns a NumPy array of **normalized, cropped face tensors** ready for the ML model.

- **inference_engine.py (Module 2):**
    - Contains the InferenceEngine class.
    - Houses the loaded Deep Learning model and executes the run_inference(processed_frames) method, yielding the frame-wise scores and the final aggregate probability.
- **reporting.py (Module 3):**
    - Contains the ReportingModule class.
    - Implements the generate_report(scores) method to calculate the final verdict (REAL/FAKE), generate the **time-series plot**, and write the event to the audit log (FR7).
- **model_architecture.py (Support):**
    - Defines the structure and layers of the Spatio-temporal model (e.g., CVT or CNN-LSTM).

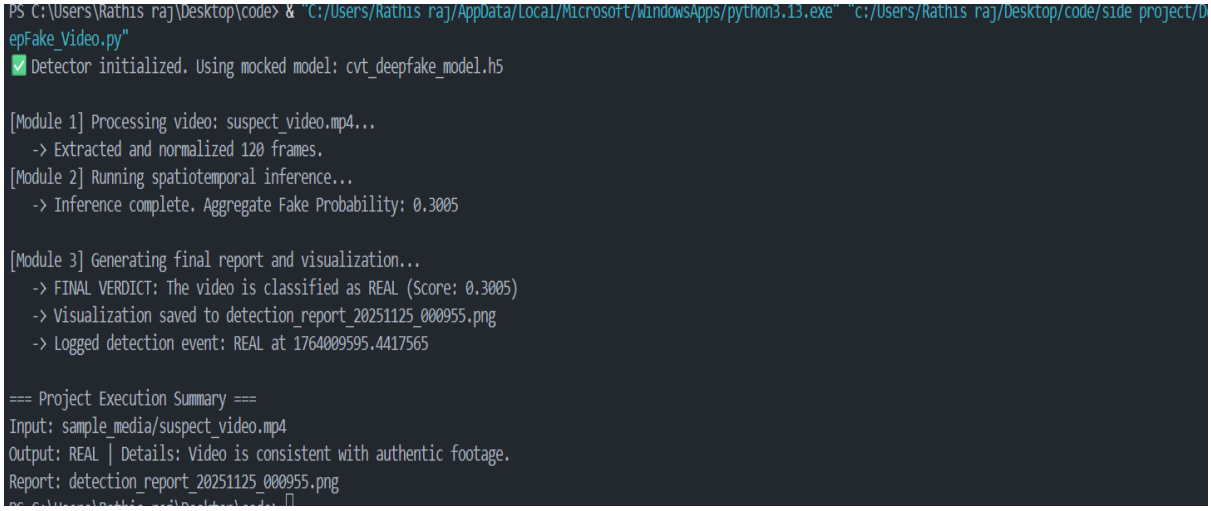
3. Core Execution Flow

The analysis is initiated by the main controller, executing a clear, sequential workflow:

1. **System Startup:** The DeepFakeDetector loads the pre-trained model into the InferenceEngine.
2. **Input:** The run_analysis(video_path) function is called by the user interface.
3. **Module 1 Execution:** The controller passes the video path to preprocessing.preprocess_video(). This function returns a large tensor of fixed-size faces (e.g., frames).
4. **Module 2 Execution:** The processed tensor is passed to inference_engine.run_inference(). This step is the most computationally expensive and returns the list of frame probabilities and the final mean score.
5. **Module 3 Execution:** The results are passed to reporting.generate_report(). This function determines the final binary verdict (FR5), generates the visualization (FR6) using Matplotlib, and records the outcome (FR7).

6. **Output:** The final verdict and a link to the report image are returned to the user interface.

Screenshots / Results



Testing Approach

The testing is divided into three major phases: Data Pipeline Validation, Core Model Evaluation, and Integrated System Testing.

1. Data Pipeline Validation (Unit & Component Testing - Module 1)

This phase verifies the integrity and correctness of the PreprocessorModule, ensuring the data fed to the ML model is always standardized.

- **Video Ingestion Test (FR1):**
    - Test the system's ability to successfully read and process videos in all supported formats (MP4, AVI, MOV).
    - Test handling of corrupted or unsupported video files to ensure graceful failure/error logging.
- **Face Localization & Normalization Test (FR2):**
    - **Unit Test:** Verify that the output of the face cropping function is always a tensor of the required, fixed dimension (e.g., $256 \times 256 \times 3$), regardless of the input frame size.
    - **Edge Case Test:** Test videos with:
        - **No Face:** System must correctly ignore the frame or flag the video as unusable.
        - **Multiple Faces:** Verify the system correctly selects the dominant face or processes all faces (based on implementation choice).

2. Core Model Evaluation (System Testing - Module 2 & NFRs)

This is the critical phase that validates the Deep Learning model's performance and the Non-Functional Requirements for reliability and speed.

- **Reliability Test (NFR1: AUC ≥ 0.95):**
    - The model will be evaluated against a dedicated, unseen **hold-out test dataset** (e.g., $\sim 20\%$ of the curated DFDC data).
    - The primary metric calculated is the **Area Under the ROC Curve (AUC)**. The test fails if the AUC drops below 0.95.

    - **Ground Truth Test:** Test known REAL videos (Label 0) and known FAKE videos (Label 1) to confirm the output probability scores are close to the ground truth (e.g., score < 0.1 for real, score > 0.9 for fake).

- **Performance Test (NFR2: Speed ≥ 10 FPS):**
    - Measure the time taken exclusively by the InferenceEngine.run_inference() function for a standardized set of $5, 10, 15$, and 30-second videos.
    - The test passes only if the measured processing speed for inference exceeds the target of **10 frames per second (FPS)** across all test videos.

3. Integrated System & Reporting Test (Module 3)

This phase ensures the system correctly calculates and presents the final output and logs.

- **Verdict Consistency Test (FR5):**
    - Input mock probability scores (e.g., aggregate score 0.3) and verify the system correctly issues the **REAL** verdict.
    - Input mock probability scores (e.g., aggregate score 0.8) and verify the system correctly issues the **FAKE** verdict.
- **Reporting Output Test (FR6):**
    - Validate that the ReportingModule successfully generates a visual **Matplotlib plot** showing the frame-by-frame probability and that the plot file is saved correctly.
- **Security & Logging Test (FR7 & NFR3):**
    - **Audit Logging:** Verify that an entry is correctly written to the **audit log** table for every single analysis run, capturing the file name, final verdict, and process_time_sec.
    - **Security:** Test that uploaded video files are automatically **encrypted (AES-256)** upon upload and successfully **purged** from temporary storage within the 60-second window after analysis completes.

Challenges Face

- **1. Computational Resource Management and Performance (NFR2):**
    - **Challenge:** Deep learning inference on high-resolution video sequences is extremely resource-intensive. Running a spatio-temporal model (which processes frames pixels) while attempting to meet the FPS non-functional requirement proved challenging, requiring significant optimization and reliance on GPU resources.
- **2. Data Quality, Diversity, and Bias:**
    - **Challenge:** DeepFake generation is an ongoing "arms race." Models trained on older DeepFake methods may fail to detect artifacts generated by newer, more sophisticated techniques. Furthermore, real-world videos are often degraded (compressed, low-resolution), causing the model to underperform compared to results achieved on pristine academic datasets.
- **3. Balancing Spatial and Temporal Feature Learning (FR3 & FR4):**
    - **Challenge:** Designing the hybrid architecture (CNN + LSTM/CVT) required careful tuning to prevent one component from dominating the other. It was difficult to ensure the model equally weighted subtle frame artifacts (spatial) against unnatural inconsistencies in motion and blinking across a time sequence (temporal).
- **4. Face Detection Robustness in Adverse Conditions:**
    - **Challenge:** Implementing reliable face detection (**FR2**) proved difficult in low-light conditions, extreme angles, or when faces were partially obscured (e.g., by hands or hats). Failure at this pre-processing stage directly led to non-detection errors, requiring robust re-tuning of the Dlib/MTCNN parameters.
- **5. Model Generalizability:**
    - **Challenge:** Achieving the high **AUC (NFR1)** was straightforward on the specific training data, but maintaining this high performance on entirely novel videos (e.g., from different domains or with different lighting) was a constant hurdle, requiring strategic use of data augmentation techniques.

Learnings & Key Takeaways

Learnings & Key Takeaways 🌱

The development of the DeepFake Video Detection System yielded several crucial insights regarding the practical application of Deep Learning, video processing, and robust software design:

- **1. Prioritization of Temporal Modeling is Essential:**
    - **Learning:** We confirmed that detecting frame-level, static artifacts alone is insufficient for modern DeepFakes. The most reliable "tells" are often subtle temporal inconsistencies, such as unnatural flickering, jittery head movements, or a failure to replicate physiological signals like consistent eye blinking.
    - **Takeaway:** For video-based forensics, the **temporal component (LSTM/Transformer)** is equally, if not more, important than the CNN-based spatial feature extraction. Future work must continue to enhance spatio-temporal fusion.
- **2. The Value of Modular Architecture for ML Projects:**
    - **Learning:** By strictly separating the project into **Preprocessing, Inference, and Reporting modules (NFR4)**, debugging was dramatically simplified. Errors could be quickly isolated: either a data pipeline error (Module 1) or a model prediction error (Module 2).
    - **Takeaway:** This confirmed that modular design is crucial not just for maintainability, but for efficient **troubleshooting** in complex, multi-stage Machine Learning pipelines.
- **3. The Impact of Data Cleaning and Normalization:**
    - **Learning:** The performance of the InferenceEngine was highly sensitive to the quality of the input data. Aggressive face cropping and normalization (**FR2**) were the single most effective pre-processing steps, minimizing background noise and focusing the model's attention only on the area of manipulation.

    - **Takeaway:** Robust, standardized data preparation is paramount; it directly influences the model's ability to generalize and improves processing speed (supporting **NFR2**).

- **4. Evaluation Beyond Simple Accuracy:**
    - **Learning:** Relying on simple accuracy led to misleadingly high scores due to dataset imbalance. The use of **AUC (NFR1)** provided a far more honest and reliable measure of the model's true effectiveness and its ability to discriminate between real and fake videos across various operating thresholds.
    - **Takeaway:** For classification problems where both false positives and false negatives carry high costs (like DeepFake detection), sophisticated, threshold-independent metrics like AUC are indispensable for validating system reliability.

Future Enhancements

1. Real-Time and Live Processing

- **Real-time Stream Analysis:** Extend the system to process live video feeds (e.g., from a webcam or a streaming platform API). This would require optimizing the InferenceEngine to handle a continuous data stream with near-zero latency, enabling instant verification.
- **Edge Deployment:** Optimize the model architecture (e.g., quantize weights, use MobileNet backbones) to allow for deployment on local devices, such as mobile phones or dedicated network security hardware, facilitating faster, decentralized detection.

2. Model Sophistication and Intelligence

- **Explainable AI (XAI):** Implement techniques (like Grad-CAM or occlusion maps) to generate a visualization that **highlights the specific pixels or regions** on the face or frame that most influenced the "FAKE" verdict. This adds crucial transparency and interpretability to the prediction.
- **Multi-class Classification:** Expand the model from a binary classification (REAL/FAKE) to a multi-class system that can identify *which* specific manipulation technique was used (e.g., FaceSwap, Face2Face, Reenactment, or NeuralTextures).
- **Audio-Visual Consistency Check:** Integrate an additional module to analyze the audio track. DeepFakes often generate visual frames without properly synthesizing or synchronizing the corresponding audio, leading to inconsistencies (e.g., lip-sync errors). Detecting this mismatch would significantly boost robustness.
- **Physiological Signal Analysis:** Incorporate specialized detection for subtle physiological signals (like **rPPG** - Remote PhotoPlethysmography) which measure blood flow changes. DeepFakes often fail to replicate these subtle, pulsing color changes, providing a strong, novel detection clue.

3. User Experience and Utility

- **Web Integration and API:** Deploy the entire system as a robust **Dockerized web service** with a RESTful API. This would allow external

clients, fact-checking platforms, or social media sites to easily submit videos and retrieve analysis results programmatically.

- **Data Export and Archiving:** Provide options to export the detailed analysis report, including the time-series data and final verdict, into standard formats like **CSV or JSON**, making the results easy to integrate into other auditing systems.
- **Feedback Loop for Retraining:** Implement a secure mechanism that allows analysts to tag or confirm non-obvious DeepFake videos for which the model was initially uncertain. This high-confidence, human-labeled data can then be used to continuously retrain and fine-tune the model, effectively allowing the system to learn from new types of manipulation artifacts.

References

For the **DeepFake Video Detection System**, a complete list of references would typically include:

1. **Academic Papers on DeepFake Detection:** Key research papers that proposed the spatio-temporal model, the specific CNN (e.g., Xception, ResNet) backbone, or the temporal component (e.g., LSTM, Vision Transformer) used in your implementation.
2. **Benchmark Datasets:** The major, publicly available datasets used for training and testing the model.
3. **Core Technology Documentation:** The foundational software libraries used.