# Deepfake Videos Detection Using Deep Learning

**A Major Project Report Submitted**

**In partial fulfillment of the requirements for the award of the degree of**

## Bachelor of Technology

## In

## Computer Science and Engineering

**By**


**ANNAM SAI SHIVANI** **21N31A0510**

**BUTTI PAVAN KUMAR** **21N31A0534**


Under the esteemed guidance of

## Dr. S. SHANTHI

**Professor, HOD-CSE**



## Department of Computer Science and Engineering

# Malla Reddy College of Engineering and Technology

**(Autonomous Institution-UGC, Govt. of India)**

(Affiliated to JNTUH, Hyderabad, Approved by AICTE, NBA&NAAC with 'A' Grade)

Maisammaguda, Kompally, Dhulapally, Secunderabad – 500100

Website: www.mrcet.ac.in


**2024 - 2025**


# Malla Reddy College of Engineering and Technology

## (Autonomous Institution-UGC, Govt. of India)

(Affiliated to JNTUH, Hyderabad, Approved by AICTE, NBA&NAAC with 'A' Grade)

Maisammaguda, Kompally, Dhulapally, Secunderabad – 500100

Website: www.mrcet.ac.in

## CERTIFICATE

This is to certify that this is the Bonafide record of the project entitle **"Deepfake Videos Detection Using Deep Learning"**, submitted by **ANNAM SAI SHIVANI (21N31A0510) and BUTTI PAVAN KUMAR (21N31A0534)** of B.Tech in the partial fulfillment of the requirements for the degree of Bachelor of Technology in Computer Science and Engineering,Department of CSE, during the year 2024-2025. The results embodied in this project report have not been submitted to any other University or Institute for the award of any degree or diploma.

| | |
|---|---|
| **Internal Guide** | **Head of the Department** |
| **Dr.S.SHANTHI** | **Dr. S. SHANTHI** |
| **Professor, HOD-CSE** | **Professor** |

**External Examiner**

# DECLARATION

We hereby declare that the project titled "**Deepfake Videos Detection Using Deep Learning"** submitted to Malla Reddy College of Engineering and Technology (UGCAutonomous), affiliated to Jawaharlal Nehru Technological University Hyderabad (JNTUH)for the award of the degree of Bachelor of Technology in Computer Science and Engineering is a result of original research carried-out in this thesis. It is further declaredthat the project report or any part thereof has not been previously submitted to any University or Institute for the award of degree or diploma.

**A Sai Shivani - (21N31A0510)**

**B Pavan Kumar - (21N31A0534)**

# ACKNOWLEDGMENT

We feel honored to place our warm salutation to our college Malla Reddy College of Engineering and Technology (UGC-Autonomous) for allowing us to do this Project as part of our B. Tech Program. We are ever grateful to our Director **Dr. V. S. K Reddy** and Principal **Dr. S. Srinivasa Rao** who enabled us tohave experience in engineering and gain profound technical knowledge.

We express our heartiest thanks to our HOD, **Dr. S. Shanthi** for encouraging usin every aspect of our course and helping us realize our full potential.

We would like to thank our Project Guide **Dr.S.Shanthi** and Project Coordinator **Ms.P.Honey Diana** for his regular guidance, suggestions, constant encouragement, continuous monitoring, and unflinching cooperation throughout project work.

We would like to thank our Class Incharge **Ms.P.Honey Diana** who despite being busy with her academic duties took time to guide and keep us on the correct path.

We would also like to thank all the faculty members and supporting staff of the Department of CSE and all other departments who have helped directly or indirectly in making our project a success.

We are extremely grateful to our parents for their blessings and prayers for the completion of our project which gave us the strength to do our project.

With regards and gratitude

**A Sai Shivani(21N31A0510)**

**Pavan Kumar(21N31A0534)**

# ABSTRACT

Deepfake videos have emerged as a critical challenge in the digital age, with their ability to manipulate and fabricate video content convincingly. These videos pose significant threats to privacy, security, and the dissemination of truthful information. Real-time detection of Deepfake videos is essential for mitigating their misuse, especially in applications requiring immediate responses, such as social media platforms, video conferencing, and live broadcasts. This project focuses on the development of a real-time Deepfake video detection framework utilizing techniques like Generative Adversarial Networks (GANs), pose a significant threat to digital media integrity, enabling misinformation, identity fraud, and security breaches. Traditional deepfake detection models struggle to generalize across different manipulation techniques due to their reliance on either spatial or temporal features. To address this challenge, we propose a hybrid deepfake detection framework integrating GenConViT (Generative Convolutional Vision Transformer), LSTM (Long Short-Term Memory), and ResNet (Residual Network) to enhance both spatial and temporal feature learning. Our approach leverages ResNet for low-level feature extraction, capturing fine-grained image details, while GenConViT, a hybrid of CNN and Vision Transformer (ViT), enhances deepfake detection by modeling both local and global dependencies in video frames. To effectively capture temporal inconsistencies across frames, LSTM is integrated, enabling the detection of subtle motion artifacts and temporal discrepancies introduced by deepfake generation methods. The framework is trained and evaluated on benchmark datasets such as FaceForensics++, DFDC, and Celeb-DF v2, achieving high accuracy and robustness against adversarial attacks.

**Keywords** : GenConViT, CNN, LSTM, RESNET, Deepfake Videos.

# TABLE OF CONTENTS

# TABLE OF CONTENTS

## LIST OF FIGURES



## LIST OF OUTPUTS

# 1. INTRODUCTION

Deepfake technology has become a major challenge in digital media authentication, enabling the creation of highly realistic yet manipulated videos. These deepfakes, generated using advanced AI techniques such as generative adversarial networks (GANs), pose serious threats in areas like misinformation, cybersecurity, and digital forensics. To address this issue, deep learning-based detection methods have emerged as a powerful solution. This project explores deepfake video detection using multiple deep learning architectures, including Convolutional Neural Networks (CNNs), Residual Networks (ResNet), Long Short-Term Memory (LSTM) networks, and the Generalized Convolutional Vision Transformer (GenConViT). CNNs are effective in extracting spatial features, allowing the detection of pixel-level artifacts in manipulated frames. ResNet, with its deep residual connections, enhances feature learning and improves accuracy in distinguishing real and fake videos. LSTM networks, designed for sequence modeling, analyze temporal inconsistencies in video frames, making them useful in identifying subtle motion distortions caused by deepfake generation. Additionally, GenConViT combines the strengths of CNNs and Vision Transformers (ViTs), leveraging self-attention mechanisms to capture fine-grained visual features and global dependencies for more precise detection. By integrating these advanced models, this project aims to develop a robust and efficient deepfake detection system capable of analyzing both spatial and temporal inconsistencies in videos. This approach enhances the reliability of digital media authentication, contributing to the fight against misinformation and the spread of manipulated content. we utilize four powerful deep learning architectures—Convolutional Neural Networks (CNNs), Residual Networks (ResNet), Long Short-Term Memory (LSTM) networks, and the Generalized Convolutional Vision Transformer (GenConViT)—to build an efficient and robust deepfake detection system. CNNs are effective in analyzing spatial features and identifying subtle inconsistencies in facial textures, lighting, and blending artifacts, which are common in deepfake videos. ResNet, an advanced version of CNNs, utilizes skip connections to enable deeper learning while mitigating the vanishing gradient problem, improving feature extraction and classification accuracy. LSTMs, a type of recurrent neural network (RNN), specialize in capturing temporal dependencies within sequential data, making them valuable for analyzing motion inconsistencies and unnatural transitions between frames in a video. Lastly, GenConViT, an advanced transformer-based model, integrates convolutional operations with self-attention mechanisms, enabling it to detect high-level

manipulations and global inconsistencies that may be missed by traditional CNNs.

## 1.1 Purpose and Objectives

The primary purpose of deepfake video detection using GenConViT is to identify and prevent the misuse of synthetic videos generated using AI. Deepfakes pose serious ethical, security, and privacy threats, and detecting them is crucial to maintaining the integrity of digital media.

- Enhancing Detection Accuracy
- Generalizing Across Different Deepfake Techniques
- Strengthening Security and Preventing Misinformation

The main objective of deepfake video detection using GenConViT is to develop a robust, efficient, and generalizable model that can accurately distinguish between real and manipulated videos. This is crucial to prevent misinformation, enhance security, and maintain digital media integrity.

**Key Objectives:**

- **Develop an Accurate Deepfake Detection System**
  Utilize CNNs, ResNet, LSTMs, and GenConViT to build a robust deepfake detection model capable of distinguishing between real and manipulated videos with high accuracy.
- **Extract and Analyze Spatial Features**
  Leverage CNNs and ResNet to identify pixel-level artifacts, inconsistencies in facial features, lighting mismatches, and blending errors introduced during deepfake generation.
- **Capture Temporal Inconsistencies**
  Employ LSTM networks to analyze motion patterns and sequential dependencies between video frames, detecting unnatural transitions and frame inconsistencies.
- **Enhance Feature Representation with Vision Transformers**
  Use GenConViT to combine convolutional feature extraction with self-attention mechanisms, enabling the model to capture fine-grained details and global dependencies.
- **Train and Evaluate Using Benchmark Datasets**
  Utilize datasets such as FaceForensics++, Deepfake Detection Challenge, and Celeb-DF to train, validate, and test the model for improved generalization across different deepfake techniques.

- **Improve Model Interpretability**
  Implement explainability techniques like Grad-CAM to visualize important regions in frames contributing to classification decisions, ensuring transparency in model predictions.
- **Optimize Model Performance for Real-World Applications**
  Fine-tune hyperparameters, apply data augmentation, and reduce computational overhead to make the model efficient for real-time and large-scale deployment in security and media verification applications.
- **Combat the Spread of Misinformation**
  Provide a reliable deepfake detection framework that can be integrated into social media platforms, news verification systems, and forensic analysis tools to prevent the distribution of manipulated content.

### 1.2.1 Existing System

Current deepfake detection systems primarily rely on deep learning techniques, including Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformer-based models. Traditional CNN-based approaches focus on analyzing spatial features in individual frames to detect inconsistencies such as unnatural facial expressions, lighting mismatches, and blending artifacts. Models like ResNet and EfficientNet have been widely used to improve feature extraction and classification accuracy. Additionally, RNNs and Long Short-Term Memory (LSTM) networks help capture temporal dependencies in videos, identifying subtle motion anomalies and inconsistencies in lip synchronization. More recently, Vision Transformers (ViTs) and hybrid models have been explored to enhance detection capabilities by leveraging self-attention mechanisms for global feature learning. Despite significant advancements, existing systems still face challenges such as generalization to unseen deepfake generation techniques, vulnerability to adversarial attacks, and high computational requirements for real-time processing. Furthermore, many detection models struggle with videos that have undergone compression, noise addition, or other post-processing effects, which degrade detection performance. To address these limitations, there is a need for more robust and efficient deepfake detection frameworks that integrate multiple deep learning architectures to improve accuracy, interpretability, and real-world applicability.

**1.GOTCHA:**

Focusing on real-time deepfake detection, GOTCHA employs a challenge-respons mechanism to establish authenticity during live video interactions. By presenting specific challenges that exploit limitations of deepfake generation pipelines, this approach effectively degrades the quality of deepfakes, facilitating their detection.

**2.GRACE (Graph-Regularized Attentive Convolutional Entanglement):**

GRACE introduces a novel approach for robust deepfake video detection by constructing a graph with sparse constraints to entangle spatial and temporal features. This method enhances stability and performance, particularly in noisy face sequences.

**3.Detect Fakes:**

A research initiative by the MIT Media Lab, Detect Fakes aims to identify AI-generated misinformation. The project focuses on uncovering subtle indicators of algorithmic manipulation in videos, enhancing the ability to detect deepfakes.

## 1.2.3 Proposed System

Deepfake Video Detection Using Generative Convolutional Vision Transformer: This project, developed by researchers Deressa Wodajo, Solomon Atnafu, and Zahid Akhtar, introduces the GenConViT model. The model combines CNN and Vision Transformer architectures for feature extraction and employs Autoencoder and Variational Autoencoder techniques to analyze latent data distributions. It has been evaluated on datasets such as DeepfakeTIMIT, achieving an average accuracy of 95.2%.

**Advantages of Proposed System**

- **Hybrid Architecture for Enhanced Feature Extraction**

  By integrating Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs), the model benefits from both local feature extraction (CNN) and global contextual learning (ViT), improving detection performance.

- **Robust to Different Deepfake Techniques**

  The use of Autoencoder (AE) and Variational Autoencoder (VAE) techniques allows the model

to analyze latent data distributions, making it capable of detecting deepfakes generated by various methods.

- **Effective in Low-Resolution and Compressed Videos**

  Many deepfake detection models struggle with low-quality or compressed videos, but GenConViT's hybrid approach improves detection performance even in such challenging scenarios.

- **Generalization Across Datasets**

  The model has been tested on standard datasets like DeepfakeTIMIT, demonstrating its ability to generalize well across different types of deepfake videos.

- **Improved Temporal and Spatial Feature Learning**

  The CNN component efficiently captures spatial artifacts in individual frames, while the transformer-based component learns temporal relationships, leading to a more comprehensive analysis of video sequences.

- **Scalability and Efficiency**

  The model leverages the strengths of transformers while maintaining computational efficiency, making it scalable for real-world applications such as media forensics, security systems, and social media platforms.

## 1.3 Scope of the Project

This project aims to develop an advanced deepfake video detection system using a combination of Convolutional Neural Networks (CNNs), Residual Networks (ResNet), Long Short-Term Memory (LSTM) networks, and the Generalized Convolutional Vision Transformer (GenConViT). The scope includes detecting deepfake videos across various datasets by analyzing both spatial and temporal inconsistencies. The model will be trained and evaluated on benchmark datasets such as DeepfakeTIMIT, FaceForensics++, and the Deepfake Detection Challenge (DFDC) to ensure its effectiveness against different deepfake generation techniques. By leveraging the strengths of CNNs and ResNet for feature extraction, LSTMs for motion analysis, and GenConViT for global attention-based learning, the system is expected to achieve high accuracy, robustness to adversarial attacks, and adaptability to new deepfake methods. Additionally, the project will focus on real-time detection

capabilities, model interpretability using techniques like Grad-CAM, and potential deployment in cybersecurity, digital forensics, and social media monitoring. As deepfake technology continues to evolve, this system will serve as a foundation for future improvements, incorporating multimodal learning and real-time detection enhancements for broader real-world applications.

**Key Features**

**Multi-Model Integration for Improved Accuracy**

- CNNs & ResNet will extract spatial features, detecting inconsistencies in textures, lighting, and facial features.
- LSTMs will analyze motion patterns and sequential dependencies, identifying unnatural movements and temporal inconsistencies in videos.
- GenConViT will enhance feature representation by combining convolutional processing with transformer-based attention mechanisms, making the model more adaptable to diverse deepfake techniques.

**Dataset Utilization & Model Training**

- The system will be trained and validated on benchmark deepfake datasets such as:
    - FaceForensics++ (high-quality and compressed fake videos)
    - DeepfakeTIMIT (synthetically generated videos)
    - Deepfake Detection Challenge (DFDC) (real-world deepfake samples)
    - Celeb-DF (low-quality, hard-to-detect fakes)
- Training will include data augmentation, adversarial learning, and fine-tuning to enhance generalization and minimize false positives.

**Real-Time and Scalable Detection**

- The system will be optimized for real-time processing with GPU acceleration, enabling deployment in real-world scenarios.
- It will focus on efficiency, allowing integration into social media platforms, video authentication tools, and forensic applications.

**Robustness Against Adversarial Attacks & Data Variations**

- Using Autoencoders (AE) and Variational Autoencoders (VAE), the model will learn latent representations to detect even subtle deepfake manipulations.
- It will address challenges posed by compressed videos, adversarial perturbations, and deepfake techniques not seen during training.

**Explainability and Interpretability**

- Grad-CAM visualization will be used to highlight the key features contributing to a classification decision, improving model transparency.
- This will be crucial for forensic investigations and legal applications where explainability is necessary.

**Future Enhancements & Research Possibilities**

- The project lays the groundwork for future improvements, such as:
    - Multimodal deepfake detection, integrating audio, facial expressions, and contextual cues.
    - Federated learning approaches for privacy-preserving deepfake detection.
    - Lightweight models for deployment on mobile and edge devices.

**Real-Time and Scalable Detection**

- The system will be optimized for real-time processing with GPU acceleration, enabling deployment in real-world scenarios.
- It will focus on efficiency, allowing integration into social media platforms, video authentication tools, and forensic applications.

By integrating deep learning, AI security, and real-time detection, this project aims to create a highly accurate, scalable, and interpretable deepfake detection framework that can be applied across various domains, including cybersecurity, media forensics, law enforcement, and digital content verification.

# 2. LITERATURE SURVEY

Deepfake detection has become a critical research area due to the increasing use of AI-generated synthetic videos for misinformation, identity fraud, and cybercrime. Several studies have explored deep learning techniques to detect deepfakes by analyzing spatial and temporal inconsistencies in manipulated videos. Early detection methods primarily relied on traditional image forensics, such as analyzing pixel artifacts, color inconsistencies, and compression traces. However, these techniques proved ineffective against advanced deepfake generation methods powered by Generative Adversarial Networks (GANs) and Autoencoders (AE).

To overcome these challenges, Convolutional Neural Networks (CNNs) have been widely used for feature extraction, detecting deepfake artifacts such as unnatural facial textures, misaligned shadows, and blending errors. Studies using ResNet and EfficientNet have shown improved accuracy by leveraging deeper architectures and skip connections to extract fine-grained features. However, CNNs often struggle with generalization across different deepfake datasets due to overfitting and a lack of temporal modeling.

**Early Approaches**

Traditional methods for forgery detection relied on image forensics techniques, such as analyzing pixel-level artifacts, color inconsistencies, and compression traces. However, these methods became less effective as deepfake generation techniques, particularly Generative Adversarial Networks (GANs) and Autoencoders (AEs), advanced, producing more realistic fake videos.

**CNN-Based Approaches**

To overcome the limitations of forensic-based methods, Convolutional Neural Networks (CNNs) were introduced for feature extraction and classification. Studies have shown that ResNet, EfficientNet, and Xception models perform well in deepfake detection by capturing fine-grained facial features, such as unnatural skin textures, misaligned shadows, and blending artifacts. The study by Afchar et al. (2018) introduced MesoNet, a CNN-based approach specifically designed for deepfake detection, focusing on shallow architectures for computational efficiency. Despite the success of CNNs, they struggle with generalization across different datasets, especially when trained on specific deepfake generation techniques.

**Temporal Modeling with RNNs and LSTMs**

Since deepfake videos involve both spatial and temporal inconsistencies, researchers incorporated Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks to analyze motion-related anomalies. Works by Sabir et al. (2019) demonstrated that LSTMs could detect unnatural facial expressions, blinking patterns, and lip-sync mismatches. Hybrid approaches combining CNNs with LSTMs improved detection performance by integrating both spatial and temporal features. However, RNN-based models often require high computational resources, limiting their applicability for real-time detection.

**Transformer-Based Models and GenConViT**

With the rise of Vision Transformers (ViTs), deepfake detection has seen significant improvements. Unlike CNNs, which focus on local features, ViTs utilize self-attention mechanisms to capture long-range dependencies across entire images. The Generative Convolutional Vision Transformer (GenConViT), proposed by Wodajo et al., introduced a hybrid approach combining CNNs and Transformers to enhance feature extraction. The model also incorporates Autoencoders (AE) and Variational Autoencoders (VAE) to analyze latent representations of deepfake manipulations. The study showed that GenConViT achieved 95.2% accuracy on the DeepfakeTIMIT dataset, outperforming traditional deepfake detection models.

# 3.SYSTEM ANALYSIS

System requirements are the functionality that is needed by a system in order to satisfy the customer's requirements. System requirements are abroad and a narrow subject that could be implemented to many items. The requirements document allows the project team to have a clear picture of what the software solution must do before selecting a vendor. Without an optimized setof future state requirements, the project team has no effective basis to choose the best system for your organization.

## 3.1 HARDWARE AND SOFTWARE REQUIREMENTS

### 3.1.1 Hardware Requirements

| | | |
|---|---|---|
| **Processor** | **:** | i5(2nd generation or later) |
| **Ram** | **:** | 4 GB or above. |
| **Hard disk** | **:** | 10GB or above. |

### 3.1.2 Software Requirements

| | | |
|---|---|---|
| **Technology/Language** | **:** | Python |
| **Operating System** | **:** | Windows |
| **IDE** | **:** | VScode |

## 3.2 SOFTWARE REQUIREMENT SPECIFICATION

Requirement specification describes what the software will do and how it will be expected to perform. It also describes the functionality the product needs to fulfil all stakeholders (business, users) needs. A software requirements specification is the basis for your entire project. It lays the framework that every team involved in development will follow. It’s used to provide critical information to multiple teams — development, quality assurance, operations, and maintenance. This keeps everyone on the same page. Using the SRS helps to ensure requirements are fulfilled. And it can also help you make decisions about your product’s lifecycle — for instance, when to retire a feature. Writing an SRS can also minimize overall development time and costs. Embedded

development teams especially benefit from using an SRS

### 3.2 SOFTWARE REQUIREMENT SPECIFICATION

Software Requirement Specification (SRS) is the starting point of the software developing activity. As system grew more complex it became evident that the goal of the entire system cannot be easily comprehended. Hence the need for the requirement phase arose. The software project is initiated by the client needs. The SRS is the means of translating the ideas of the minds of clients (the input) into a formal document (the output of the requirement phase.) The SRS phase consists of two basic activities: 1) Problem/Requirement Analysis: The process is order and more nebulous of the two, deals with understand the problem, the goal and constraints. 2) Requirement Specification: Here, the focus is on specifying what has been found giving analysis such as representation, specification languages and tools, and checking the specifications are addressed during this activity. The Requirement phase terminates with the production of the validate SRS document. Producing the SRS document is the basic goal of this phase.

**Role of SRS:**

The purpose of the Software Requirement Specification is to reduce the communication gap between the clients and the developers. Software Requirement Specification is the medium through which the client and user needs are accurately specified. It forms the basis of software development. A good SRS should satisfy all the parties involved in the system.

#### 3.2.1 Functional Requirements

The functional requirements define the specific operations and functionalities that the Deepfake Video Detection System should perform. The system will leverage CNN, ResNet, LSTM, and GenConViT to detect manipulated videos efficiently.

**1. Data Input & Preprocessing**

- The system should accept video inputs in formats such as MP4, AVI, and MOV.
- Frame extraction should be performed to convert videos into individual frames for processing.
- Preprocessing should include resizing, normalization, and data augmentation to improve

model generalization.

- Support for both real-time webcam-based detection and batch processing of videos.

**2. Feature Extraction & Model Processing**

- CNN and ResNet should extract spatial features such as artifacts, texture inconsistencies, and blending errors.
- LSTM should analyze temporal inconsistencies, including unnatural movements and blinking patterns.
- GenConViT should leverage Transformer-based attention mechanisms to improve feature representation and enhance detection accuracy.
- Autoencoder (AE) and Variational Autoencoder (VAE) should analyze latent data representations for better classification of real vs. fake videos.

**3. Deepfake Classification**

- The system should classify videos into "Real" or "Fake" based on the extracted features.
- It should provide a confidence score (%) indicating the likelihood of a video being fake.
- Support for multi-class classification (e.g., different deepfake generation techniques like FaceSwap, DeepFaceLab, etc.).

**4. Model Training & Evaluation**

- The system should allow training and fine-tuning on datasets like FaceForensics++, DeepfakeTIMIT, DFDC, and Celeb-DF.
- Support for cross-validation and hyperparameter tuning to improve performance.
- Performance metrics such as accuracy, precision, recall, F1-score, and AUC-ROC should be computed.
- The system should provide Grad-CAM visualizations to interpret model predictions.

**5. Real-Time Deepfake Detection**

- The system should support real-time detection via webcam input.
- Optimization techniques should be used to ensure low latency and high FPS (frames per second) processing.

- GPU acceleration should be utilized for efficient deep learning inference.

**6. User Interface & Output Generation**

- The system should have a user-friendly GUI or web interface for video upload and analysis.
- The results should include:
    - Classification Label: Real or Fake
    - Confidence Score (%)
    - Highlighted Regions with Artifacts (Grad-CAM Visualization)
- The system should generate a detailed report (PDF/CSV) summarizing the analysis.

**7. Security & Robustness**

- The model should be resistant to adversarial attacks (e.g., subtle noise perturbations).
- It should handle compressed and low-resolution videos effectively.
- The system should log and store detection results for audit and forensic analysis.

**8. Deployment & Scalability**

- The system should support cloud-based deployment for large-scale deepfake detection.
- Lightweight models should be available for mobile and edge-device deployment.
- Support for API integration, allowing external applications to use deepfake detection services.

### 3.2.2 Non-Functional Requirements

Non-functional requirements, often referred to as quality attributes or constraints, define the overarching qualities or characteristics that a system must possess, rather than specifying particular behaviors or functionalities. These requirements focus on the overall performance, usability, security, and other qualities that contribute to the system's effectiveness and user satisfaction. Unlike functional requirements that detail specific features, non-functional requirements provide criteria for evaluating the system's performance and behavior under different conditions. Examples of non-functional requirements include scalability, reliability, security, usability, performance, and maintainability. Addressing non-functional requirements is crucial for ensuring that the system meets the broader expectations and constraints set by stakeholders.

Below are the non-functional requirements for the Machine learning adopted potability assessment for safe drinking water.

**1) Performance:** Ensure that the system processes video frames and gestures in real-time. The gesture recognition and classification should occur within milliseconds to provide instant feedback. Optimize the Random Forest classifier for speed and implement parallel processing where necessary to minimize latency.

**2) Usability:** Design a user-friendly interface that allows users to easily view the real-time video feed, detected gestures, and translated text. The interface should be intuitive, with clear navigation and feedback for successful or failed gesture recognition.

**3) Reliability:** The system should maintain stable performance under varying conditions, such as different lighting environments and hand movement speeds. Incorporate error handling to manage misclassifications and provide users with accurate feedback**.**

**4) Security:** Implement security measures to ensure that video data is not stored or shared without user consent. Any sensitive data (e.g., if user profiles or saved gestures are involved) should be encrypted during transmission and storage.

**5) Scalability:** The system should be scalable to handle an increasing number of gestures and users. As more gestures are added, the classifier should still perform efficiently, and the architecture should.

# 4. SYSTEM DESIGN

## 4.1 DESCRIPTION

System design is the process of creating a system's architecture, parts, and interfaces to ensure that it satisfies the needs of its users.Thus, in order to examine the design of this project, we first go through the specifics of establishingthe concept of drone detection through a few fundamental modules that would clearly describe theworkings of the system that would come from the development.

System design is transition from a user oriented document to programmers or data base personnal. The design is a solution, how to approach to the creation of a new system. This is composed of several steps. It provides the understanding and procedural details necessary for implementing the system recommended in the feasibility study. Designing goes through logical and physical stages of development, logical design reviews the present physical system, prepare input and output specification, details of implementation plan and prepare a logical design walkthrough.

## 4.1 Software Design:

In designing the software following principles are followed:

- **Modularity and Partitioning**: Software is designed such that, each system should consists of hierarchy of modules and serve to partition into separate function.

- **Coupling**: Modules should have little dependence on other modules of a system. Cohesion: Modules should carry out in a single processing function.

- **Shared use**: Avoid duplication by allowing a single module be called by other that need the function it provides.

## 4.2 Architecture Diagram

Architecture diagram is a diagram of a system, in which the principal parts or functions are represented by blocks connected by lines that show the relationships of the blocks. The block diagram is typically used for a higher level, less detailed description aimed more at understanding the overall concepts and less at understanding the details of implementation.

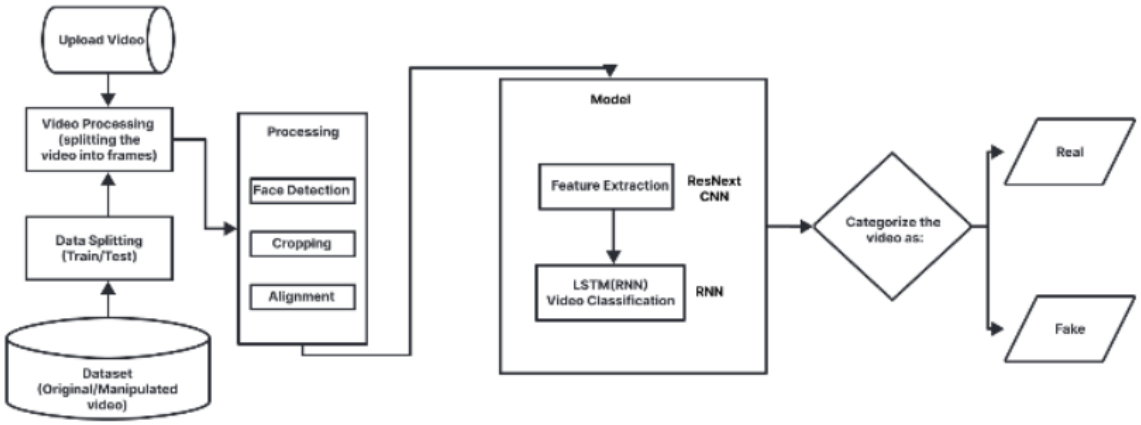



Fig – System Architecture

The Deepfake Video Detection Pipeline follows a structured approach to identify manipulated videos using ResNext CNN and LSTM (RNN) for feature extraction and classification. The process begins with the upload of a video, which is then preprocessed by splitting it into individual frames to analyze each frame separately. The dataset, containing both real and manipulated videos, is divided into training and testing sets to train the model effectively. Preprocessing techniques such as face detection, cropping, and alignment ensure that the system focuses only on facial regions, improving detection accuracy.

Once the preprocessing is complete, the system extracts spatial features using ResNext CNN, which helps identify texture inconsistencies, blending errors, and unnatural artifacts in frames. Since deepfake artifacts may also appear over time, an LSTM-based RNN is employed to analyze temporal inconsistencies across frames, capturing unnatural facial expressions, blinking patterns, and movement anomalies. After processing through the model, the system categorizes the video as either real (authentic) or fake (manipulated). This approach ensures a robust deepfake detection mechanism by combining spatial and temporal feature analysis,

making it effective for applications in media forensics, cybersecurity, and fake content verification.

## 4.3 UML DIAGRAMS

The unified modeling is a standard language for specifying, visualizing, constructing and documenting the system and its components is a graphical language which provides a vocabulary and set of semantics and rules. The UML focuses on the conceptual and physical representation of the system. It captures the decisions and understandings about systems that must be constructed. It is used to understand, design, configure and control information about the systems.

Depending on the development culture, some of these artifacts are treated more or less formally than others. Such artifacts are not only the deliverables of a project; they are also critical in controlling, measuring, and communicating about a system during its development and after its deployment.

The UML addresses the documentation of a system's architecture and all of its details. The UML also provides a language for expressing requirements and for tests. Finally, the UML provides a language for modeling the activities of project planning and release management.

**BUILDING BLOCKS OF UML:**

The vocabulary of the UML encompasses three kinds of building blocks:

- ✓ Things.
- ✓ Relationships.
- ✓ Diagrams.

**Things in the UML:**

Things are the abstractions that are first-class citizens in a model; relationships tie these things together; diagrams group interesting collections of things.

There are four kinds of things in the UML:

- Structural things.
- Behavioral things.
- Grouping things
- Annotational things

**1.Structural things** are the nouns of UML models. The structural things used in the project design are:

- ✓ First, a **class** is a description of a set of objects that share the same attributes, operations, relationships and semantics.

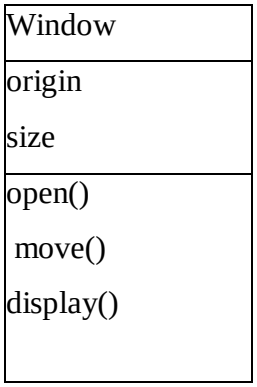



**Fig: Classes**

- ✓ Second, a **use case** is a description of set of sequence of actions that a system performs that yields an observable result of value to particular actor.

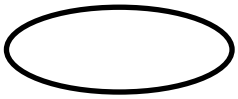


**Fig: Use Cases**

- ✓ Third, a node is a physical element that exists at runtime and represents a computational resource, generally having at least some memory and often

processing capability.

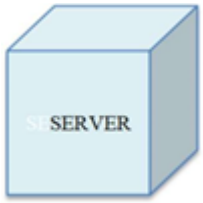



**Fig: Node**

2. **Behavioral things** are the dynamic parts of UML models. The behavioral thing used Is:

   ✓ Interaction: An interaction is a behavior that comprises a set of messages exchanged among a set of objects within a particular context to accomplish a specific purpose. An interaction involves a number of other elements, including messages, action sequences (the behavior invoked by a message, and links (the connection between objects).


**Fig: Messages**

**Relationships in the UML:**

There are four kinds of relationships in the UML:

- Dependency.
- Association.
- Generalization.
- Realization.

- A **dependency** is a semantic relationship between two things in which a change to one thing may affect the semantics of the other thing (the dependent thing).



**Fig: Dependencies**

- An **association** is a structural relationship that describes a set links, a link being a connection among objects. Aggregation is a special kind of association, representing a structural relationship between a whole and its parts.



**Fig: Association**

- A **generalization** is a specialization/ generalization relationship in which objects of thespecialized element (the child) are substitutable for objects of the generalized element(the parent).



**Fig: Generalization**

- A **realization** is a semantic relationship between classifiers, where in one classifier
- specifies a contract that another classifier guarantees to carry out.



**Fig: Realization**

### 4.3.1 UML DIAGRAMS:

#### 1. Data Flow Diagram

A data-flow diagram is a visual representation of how data moves through a system or a process (usually an information system). The data flow diagram also shows the inputs and outputs of each entity as well as the process itself. A data-flow diagram lacks control flow, loops, and decision-making processes. With a flowchart, certain operations based on the data can be depicted. The flowchart can be used to understand how the data flows in this project. A video clip from a camera is used as the input in this case, and the number of frames on which the algorithm operateswill later be converted.

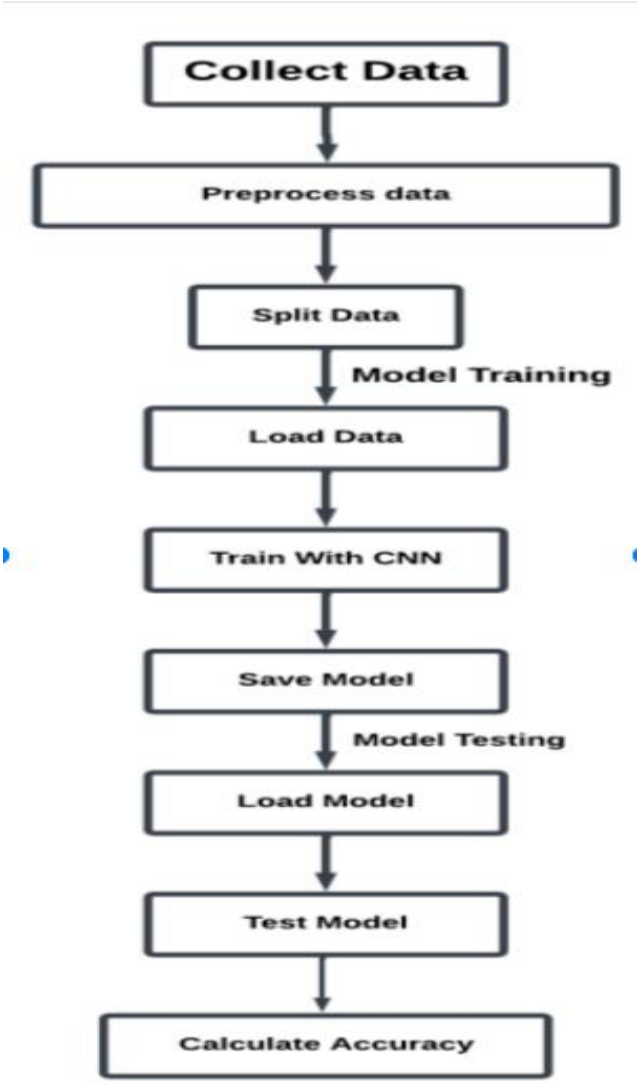



**Fig. Data Flow Diagram**

The given data flow diagram represents the workflow of a deepfake video detection model using a Convolutional Neural Network (CNN). It outlines the steps from data collection to model evaluation, ensuring a structured approach to training and testing the model. Here's a step-by-step explanation:

1. Collect Data – The first step involves gathering a dataset consisting of both real and manipulated (deepfake) videos to train the model effectively.
2. Preprocess Data – The collected data is processed by converting videos into frames, detecting faces, normalizing image sizes, and applying necessary augmentations.
3. Split Data – The dataset is divided into training and testing sets to train the model and evaluate its performance.
4. Model Training Phase:
    - Load Data – The preprocessed data is fed into the model.
    - Train with CNN – The Convolutional Neural Network (CNN) is used to extract features from the images and learn patterns to distinguish between real and fake videos.
5. Save Model – Once the CNN is trained, the trained model is saved for further testing and deployment.
6. Model Testing Phase:
    - Load Model – The saved trained model is loaded to test its performance on new data.
    - Test Model – The model is tested using unseen test data to evaluate its effectiveness in detecting deepfake videos.
7. Calculate Accuracy – The final step involves computing the model's accuracy, which helps determine how well the CNN performs in detecting real and fake videos.

## 2. Use Case Diagram

To depict a system's dynamic behavior, use case diagrams are often employed. Using use cases, actors, and their interactions, it captures the functionality of the system. A system or subsystem of an application's necessary duties, services, and operations are modelled. It shows a system's high-level functionality as well as how a user interacts with that system.

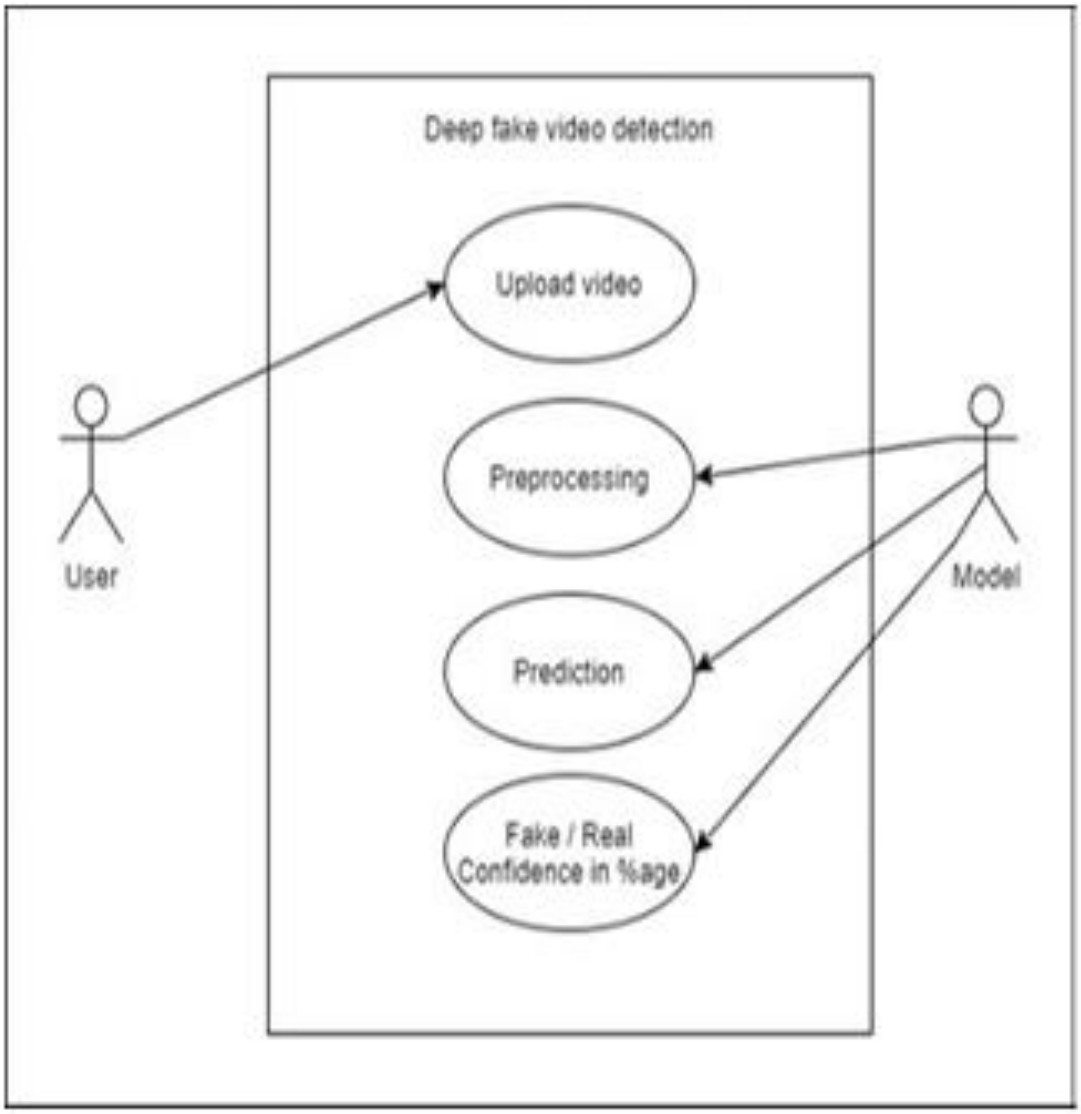



**Fig. Use Case Diagram**

The Use Case Diagram for deepfake video detection illustrates the interaction between the user and the detection model. The process begins with the user uploading a video, which serves as input for the system. Once uploaded, the preprocessing step takes place, where the model extracts key features by performing frame extraction, face detection, alignment, and cropping. After

preprocessing, the model proceeds to the prediction stage, where it analyzes the video frames to determine whether the video is real or deepfake using deep learning techniques such as CNNs, LSTMs, and Vision Transformers. Finally, the system provides the output in terms of classification (Real/Fake) along with a confidence score in percentage, indicating the model's certainty in its decision. This structured approach ensures an efficient and automated process for detecting manipulated videos, making it useful for applications in cybersecurity, media authentication, and fake content identification.

**Elements in the Diagram**

1. **Actors**:
    - **User**: The individual who uploads the video for deepfake detection.
    - **Model**: The deep learning model responsible for processing and classifying the video.
2. **Processes (Use Cases)**:
    - **Upload Video**: The user uploads a video for analysis.
    - **Preprocessing**: The model processes the video by extracting frames and normalizing them.
    - **Prediction**: The model classifies the video as **fake or real**.
    - **Fake/Real Confidence in %**: The model outputs the classification result along with a confidence score.

**Explanation of Workflow**

1. **The user uploads a video** into the deepfake detection system.
2. **Preprocessing** is performed to extract relevant features from frames (handled by the model).
3. **Prediction** is made by the model, classifying the video as real or deepfake.
4. **Final output** is provided as a classification result with a confidence percentage.

This diagram visually represents the interaction between users and the model within a deepfake detection system.

## 3.Activity Diagram

An activity diagram is a type of UML (Unified Modeling Language) diagram used to model the workflow of a system or process. It focuses on the sequence of activities and the flow of control from one activity to another. They provide a detailed view of how processes are executed, including the various steps involved, the order in which they occur, and how different branches of the process interact.

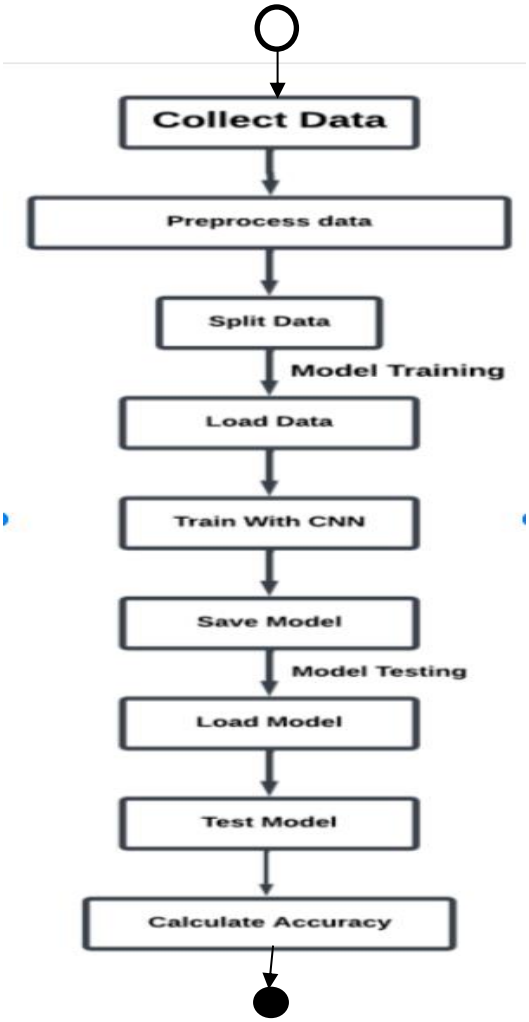



**Fig - Activity Diagram**

Here's a step-by-step explanation:

8. Collect Data – The first step involves gathering a dataset consisting of both real and manipulated (deepfake) videos to train the model effectively.
9. Preprocess Data – The collected data is processed by converting videos into frames, detecting faces, normalizing image sizes, and applying necessary augmentations.
10. Split Data – The dataset is divided into training and testing sets to train the model and evaluate its performance.

11. Model Training Phase:

- Load Data – The preprocessed data is fed into the model.
- Train with CNN – The Convolutional Neural Network (CNN) is used to extract features from the images and learn patterns to distinguish between real and fake videos.

12. Save Model – Once the CNN is trained, the trained model is saved for further testing and deployment.

13. Model Testing Phase:

- Load Model – The saved trained model is loaded to test its performance on new data.
- Test Model – The model is tested using unseen test data to evaluate its effectiveness in detecting deepfake videos.

14. Calculate Accuracy – The final step involves computing the model's accuracy, which helps determine how well the CNN performs in detecting real and fake videos.

## 4.Sequence Diagram

In UML, sequence diagrams display how and in what order certain items interact with one another. It's crucial to remember that they depict the interactions for a certain circumstance. The interactions are depicted as arrows, while the processes are portrayed vertically. The objective of sequence diagrams and their fundamentals are explained in this article. To understand more about sequence diagrams, you may also look at this comprehensive tutorial.

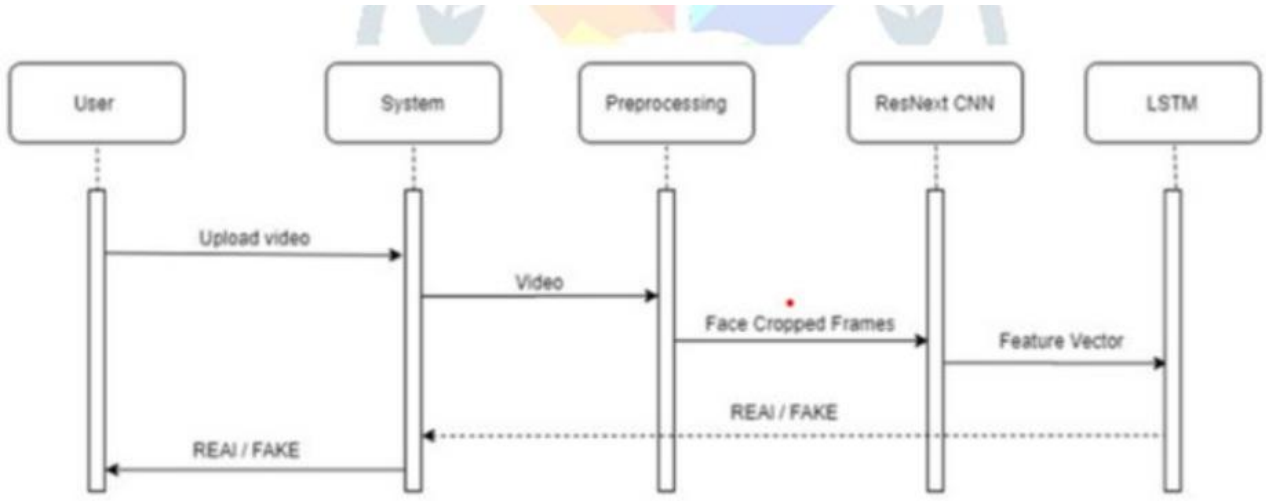



**Fig-Sequence Diagram**

The sequence diagram shows the interaction between the user, the tour package, the payment system, and the booking system in an online tourism management system.

The user starts by viewing available tour packages. The tour package system responds with a list of available packages. The user then selects a package and books it.

The booking system then requests payment from the payment system. The payment system confirms the payment and the booking system confirms the booking.

The diagram illustrates the following steps:

**User Uploads Video** – The process starts when the user uploads a video to the system for verification.

**System Processing** – The system receives the video and passes it to the preprocessing module.

**Preprocessing Stage** – This stage involves face detection, cropping, and frame extraction from the uploaded video to focus on relevant facial features.

**Feature Extraction Using ResNext CNN** – The face-cropped frames are sent to a ResNext CNN model, which extracts deep feature representations from the images. This helps in identifying manipulated patterns in the video frames.

**Temporal Analysis Using LSTM** – The extracted feature vectors are then passed to an LSTM (Long Short-Term Memory) model, which analyzes the temporal relationships between frames, helping in detecting subtle changes over time.

**Final Classification** – Based on the extracted features and temporal analysis, the system classifies the video as Real or Fake, and the result is returned to the user.

A sequence diagram is crucial in software development, especially for a Deepfake Video Detection System, because it helps in visualizing the step-by-step interaction between system components over time.

# 5. METHODOLOGY

## 5.1 TECHNOLOGIES USED

The deepfake video detection system leverages a combination of deep learning, computer vision, and AI-based models to accurately classify manipulated videos. The core technology stack includes Python as the primary programming language, along with deep learning frameworks such as TensorFlow and PyTorch for model development and training. The GenConViT (Generative Convolutional Vision Transformer) model integrates convolutional and transformer architectures, enabling it to extract both local and global video features. Additionally, ResNet is employed for spatial feature extraction from individual video frames, while LSTM (Long Short-Term Memory) networks capture temporal dependencies across frames, helping to detect subtle inconsistencies in motion.

For dataset handling and preprocessing, libraries like OpenCV (for video processing), NumPy, and Pandas are utilized. The system is trained on large-scale deepfake datasets such as FaceForensics++, DFDC (Deepfake Detection Challenge), and Celeb-DF v2. The backend infrastructure can be deployed using Flask or FastAPI to serve the deepfake detection model as a web-based API. Additionally, CUDA-enabled GPUs (e.g., NVIDIA GPUs) and cuDNN acceleration are leveraged to speed up model training and inference. For evaluation and visualization, Matplotlib and Seaborn are used to analyze model performance, while metrics like accuracy, AUC, precision, and recall are computed using Scikit-learn. This combination of technologies ensures a scalable, efficient, and high-performance deepfake detection system capable of real-time analysis and classification.

The Deepfake Video Detection system utilizes a combination of advanced deep learning and computer vision technologies to effectively analyze and classify videos. Below are the key technologies used in the project:

**1. Deep Learning Frameworks**

- **TensorFlow / PyTorch** – Used for building and training deep learning models.
- **Keras** – High-level API for rapid prototyping of neural networks.

**2. Machine Learning Models**

- **CNN (Convolutional Neural Networks)** – Extracts spatial features from video frames.
- **ResNet / ResNext** – Advanced CNN architectures that enhance feature extraction.

- **LSTM (Long Short-Term Memory Networks)** – Captures temporal dependencies in video sequences.
- **Vision Transformer (ViT)** – Utilized for image analysis using self-attention mechanisms.
- **GenConViT (Generative Convolutional Vision Transformer)** – A hybrid model combining CNN and ViT for superior deepfake detection.

**3. Image and Video Processing**

- **OpenCV** – Used for video frame extraction, face detection, and preprocessing.
- **Dlib** – Face alignment and landmark detection.
- **FFmpeg** – Handles video processing tasks like frame extraction and format conversion.

**4. Dataset Management & Preprocessing**

- **Pandas & NumPy** – For handling large-scale datasets efficiently.
- **Scikit-learn** – Used for data preprocessing, splitting, and evaluation.
- **Matplotlib & Seaborn** – For visualizing results and performance metrics.

**5. Deployment & Backend Technologies**

- **Flask / FastAPI** – To develop a web-based application for real-time deepfake detection.
- **Django** – Alternative backend framework for web application deployment.

**6. Hardware & Cloud Support**

- **GPUs (NVIDIA CUDA, TensorRT)** – Accelerates deep learning model training and inference.
- **Google Colab / Kaggle Notebooks** – Provides cloud-based model training capabilities.
- **AWS / Google Cloud / Azure** – Cloud-based storage and computation for large-scale deployment.

**7. Computer Vision Libraries & Frameworks**

- **OpenCV:** Used for frame extraction, face detection, and preprocessing.
- **Dlib:** Assists in face detection and feature mapping.

- **TensorFlow & PyTorch:** Provide deep learning frameworks for training and deploying deepfake detection models.

**8. Feature Extraction Techniques**

- **Histogram of Oriented Gradients (HOG):** Helps identify texture-based differences between real and fake faces.
- **Optical Flow Analysis:** Tracks motion inconsistencies across frames.
- **Frequency Domain Analysis:** Detects artifacts in fake videos using Fourier Transform-based techniques.

**9. GAN-Based Detection Models**

- Since deepfakes are often generated using **Generative Adversarial Networks (GANs)**, specialized **GAN fingerprinting techniques** analyze the unique patterns left by different GAN architectures.
- **StyleGAN & DeepFakeGAN Detection:** Models trained to recognize artifacts introduced by deepfake generators.

**10. Audio-Visual Synchronization Analysis**

- **Lip-Sync Analysis:** Detects mismatches between speech and lip movements using LSTM-based classifiers.
- **Voice Deepfake Detection:** Uses spectral analysis and deep learning to distinguish between real and AI-generated voices.

**11. Blockchain & Digital Watermarking**

- **Blockchain-Based Video Verification:** Ensures video authenticity by maintaining an immutable record of video origins.
- **Deep Learning-Based Watermarking:** Embeds and detects invisible watermarks in videos to prevent manipulation.

**12. Cloud-Based & Edge AI Detection**

- **Cloud AI Services:** Google's DeepFake Detection API and Microsoft's Video Authenticator help verify video authenticity.
- **Edge AI Models:** Optimized lightweight deepfake detectors for real-time verification on mobile devices and embedded systems.

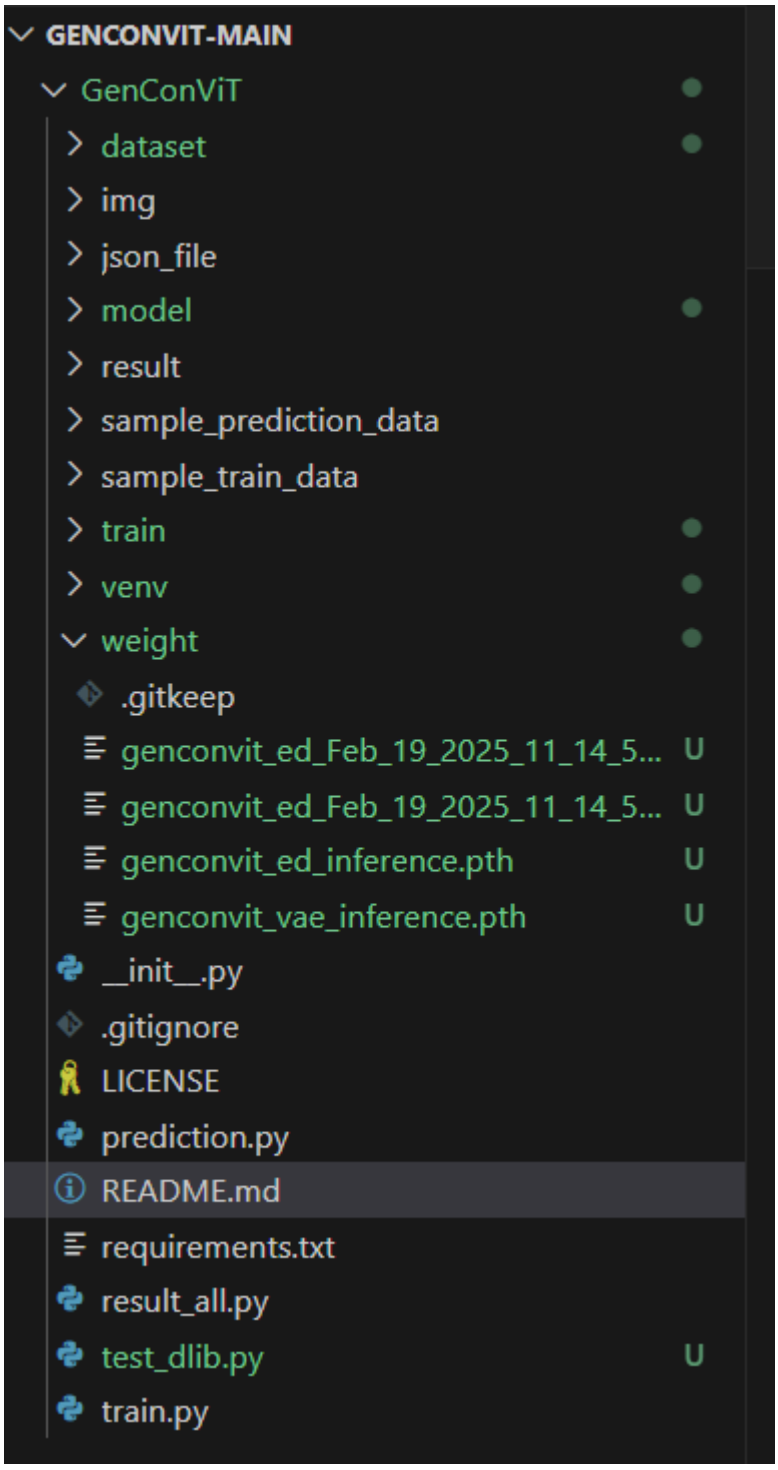
## 5.2 Libaries Used

The deepfake video detection system relies on various libraries for **data processing, deep learning, model evaluation, and deployment**. Below are the key libraries used in the project:

1. **NumPy** – For numerical operations and handling multi-dimensional arrays.
2. **Pandas** – Used for data manipulation and managing datasets.
3. **OpenCV** – Essential for video processing tasks like frame extraction, resizing, and enhancement.
4. **TensorFlow & PyTorch** – Core deep learning frameworks for building, training, and deploying the model.
5. **torchvision** – Provides pre-trained models (like ResNet) and image processing utilities.
6. **Transformers (Hugging Face)** – Useful for implementing Vision Transformers (ViTs) and GenConViT models.
7. **Keras** – High-level API for deep learning, simplifying model training and evaluation.
8. **Scikit-learn** – Offers evaluation metrics like accuracy, precision, recall, and AUC.
9. **Matplotlib & Seaborn** – Used for data visualization, training curve plotting, and model performance analysis.
10. **Albumentations** – Enhances dataset quality by applying image augmentation techniques.
11. **Flask/FastAPI** – Helps deploy the deepfake detection model as a web API for real-time inference.
12. **CUDA & cuDNN** – Accelerate deep learning computations using NVIDIA GPUs.

# 6. IMPLEMENTATION

## 6.1 FILE STRUCTURE



## 6.1 Sample Code

After completing the literature survey, we defined the system design. Based on this system design, we have selected the technologies to be used. With the help of the defined technologies, we have developed the following sample code.

```
from django.shortcuts import render, redirect
import torch
import torchvision
from torchvision import transforms, models
from torch.utils.data import DataLoader
from torch.utils.data.dataset import Dataset
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
import face_recognition
from torch.autograd import Variable
import time
import sys
from torch import nn
import json
import glob
import copy
from torchvision import models
import shutil
from PIL import Image as pImage
import time
from django.conf import settings
from .forms import VideoUploadForm

index_template_name = 'index.html'
predict_template_name = 'predict.html'
about_template_name = "about.html"
```

```
im_size = 112
mean=[0.485, 0.456, 0.406]
std=[0.229, 0.224, 0.225]
sm = nn.Softmax()
inv_normalize =  transforms.Normalize(mean=-1*np.divide(mean,std),std=np.divide([1,1,1],std))
if torch.cuda.is_available():
    device = 'gpu'
else:
    device = 'cpu'

train_transforms = transforms.Compose([
                                transforms.ToPILImage(),
                                transforms.Resize((im_size,im_size)),
                                transforms.ToTensor(),
                                transforms.Normalize(mean,std)])

class Model(nn.Module):

    def __init__(self, num_classes,latent_dim= 2048, lstm_layers=1 , hidden_dim = 2048,
bidirectional = False):
        super(Model, self).__init__()
        model = models.resnext50_32x4d(pretrained = True)
        self.model = nn.Sequential(*list(model.children())[:-2])
        self.lstm = nn.LSTM(latent_dim,hidden_dim, lstm_layers,  bidirectional)
        self.relu = nn.LeakyReLU()
        self.dp = nn.Dropout(0.4)
        self.linear1 = nn.Linear(2048,num_classes)
        self.avgpool = nn.AdaptiveAvgPool2d(1)

    def forward(self, x):
        batch_size,seq_length, c, h, w = x.shape
        x = x.view(batch_size * seq_length, c, h, w)
        fmap = self.model(x)
        x = self.avgpool(fmap)
```

```
        x = x.view(batch_size,seq_length,2048)
        x_lstm,_ = self.lstm(x,None)
        return fmap,self.dp(self.linear1(x_lstm[:,-1,:]))

class validation_dataset(Dataset):
    def __init__(self,video_names,sequence_length=60,transform = None):
        self.video_names = video_names
        self.transform = transform
        self.count = sequence_length

    def __len__(self):
        return len(self.video_names)

    def __getitem__(self,idx):
        video_path = self.video_names[idx]
        frames = []
        a = int(100/self.count)
        first_frame = np.random.randint(0,a)
        for i,frame in enumerate(self.frame_extract(video_path)):
            #if(i % a == first_frame):
            faces = face_recognition.face_locations(frame)
            try:
              top,right,bottom,left = faces[0]
              frame = frame[top:bottom,left:right,:]
            except:
              pass
            frames.append(self.transform(frame))
            if(len(frames) == self.count):
                break
        """
        for i,frame in enumerate(self.frame_extract(video_path)):
            if(i % a == first_frame):
                frames.append(self.transform(frame))
        """
```

```
        # if(len(frames)<self.count):
        #   for i in range(self.count-len(frames)):
        #         frames.append(self.transform(frame))
        #print("no of frames", self.count)
        frames = torch.stack(frames)
        frames = frames[:self.count]
        return frames.unsqueeze(0)

    def frame_extract(self,path):
      vidObj = cv2.VideoCapture(path)
      success = 1
      while success:
          success, image = vidObj.read()
          if success:
              yield image

def im_convert(tensor, video_file_name):
    """ Display a tensor as an image. """
    image = tensor.to("cpu").clone().detach()
    image = image.squeeze()
    image = inv_normalize(image)
    image = image.numpy()
    image = image.transpose(1,2,0)
    image = image.clip(0, 1)
    # This image is not used
    # cv2.imwrite(os.path.join(settings.PROJECT_DIR, 'uploaded_images',
video_file_name+'_convert_2.png'),image*255)
    return image

def im_plot(tensor):
    image = tensor.cpu().numpy().transpose(1,2,0)
    b,g,r = cv2.split(image)
    image = cv2.merge((r,g,b))
    image = image*[0.22803, 0.22145, 0.216989] +  [0.43216, 0.394666, 0.37645]
```

```
    image = image*255.0
    plt.imshow(image.astype('uint8'))
    plt.show()

def predict(model,img,path = './', video_file_name=""):
  fmap,logits = model(img.to(device))
  img = im_convert(img[:,-1,:,:,:], video_file_name)
  params = list(model.parameters())
  weight_softmax = model.linear1.weight.detach().cpu().numpy()
  logits = sm(logits)
  _,prediction = torch.max(logits,1)
  confidence = logits[:,int(prediction.item())].item()*100
  print('confidence of prediction:',logits[:,int(prediction.item())].item()*100)
  return [int(prediction.item()),confidence]

def plot_heat_map(i, model, img, path = './', video_file_name=''):
  fmap,logits = model(img.to(device))
  params = list(model.parameters())
  weight_softmax = model.linear1.weight.detach().cpu().numpy()
  logits = sm(logits)
  _,prediction = torch.max(logits,1)
  idx = np.argmax(logits.detach().cpu().numpy())
  bz, nc, h, w = fmap.shape
  #out = np.dot(fmap[-1].detach().cpu().numpy().reshape((nc, h*w)).T,weight_softmax[idx,:].T)
  out = np.dot(fmap[i].detach().cpu().numpy().reshape((nc, h*w)).T,weight_softmax[idx,:].T)
  predict = out.reshape(h,w)
  predict = predict - np.min(predict)
  predict_img = predict / np.max(predict)
  predict_img = np.uint8(255*predict_img)
  out = cv2.resize(predict_img, (im_size,im_size))
  heatmap = cv2.applyColorMap(out, cv2.COLORMAP_JET)
  img = im_convert(img[:,-1,:,:,:], video_file_name)
  result = heatmap * 0.5 + img*0.8*255
  # Saving heatmap - Start
```

```
  heatmap_name = video_file_name+"_heatmap_"+str(i)+".png"
  image_name = os.path.join(settings.PROJECT_DIR, 'uploaded_images', heatmap_name)
  cv2.imwrite(image_name,result)
  # Saving heatmap - End
  result1 = heatmap * 0.5/255 + img*0.8
  r,g,b = cv2.split(result1)
  result1 = cv2.merge((r,g,b))
  return image_name

# Model Selection
def get_accurate_model(sequence_length):
    model_name = []
    sequence_model = []
    final_model = ""
    list_models = glob.glob(os.path.join(settings.PROJECT_DIR, "models", "*.pt"))

    for model_path in list_models:
        model_name.append(os.path.basename(model_path))

    for model_filename in model_name:
        try:
            seq = model_filename.split("_")[3]
            if int(seq) == sequence_length:
                sequence_model.append(model_filename)
        except IndexError:
            pass  # Handle cases where the filename format doesn't match expected

    if len(sequence_model) > 1:
        accuracy = []
        for filename in sequence_model:
            acc = filename.split("_")[1]
            accuracy.append(acc)  # Convert accuracy to float for proper comparison
        max_index = accuracy.index(max(accuracy))
        final_model = os.path.join(settings.PROJECT_DIR, "models", sequence_model[max_index])
```

```
    elif len(sequence_model) == 1:
        final_model = os.path.join(settings.PROJECT_DIR, "models", sequence_model[0])
    else:
        print("No model found for the specified sequence length.")  # Handle no models found case

    return final_model

ALLOWED_VIDEO_EXTENSIONS = set(['mp4','gif','webm','avi','3gp','wmv','flv','mkv'])

def allowed_video_file(filename):
    #print("filename" ,filename.rsplit('.',1)[1].lower())
    if (filename.rsplit('.',1)[1].lower() in ALLOWED_VIDEO_EXTENSIONS):
        return True
    else:
        return False
def index(request):
    if request.method == 'GET':
        video_upload_form = VideoUploadForm()
        if 'file_name' in request.session:
            del request.session['file_name']
        if 'preprocessed_images' in request.session:
            del request.session['preprocessed_images']
        if 'faces_cropped_images' in request.session:
            del request.session['faces_cropped_images']
        return render(request, index_template_name, {"form": video_upload_form})
    else:
        video_upload_form = VideoUploadForm(request.POST, request.FILES)
        if video_upload_form.is_valid():
            video_file = video_upload_form.cleaned_data['upload_video_file']
            video_file_ext = video_file.name.split('.')[-1]
            sequence_length = video_upload_form.cleaned_data['sequence_length']
            video_content_type = video_file.content_type.split('/')[0]
            if video_content_type in settings.CONTENT_TYPES:
                if video_file.size > int(settings.MAX_UPLOAD_SIZE):
```

```
                video_upload_form.add_error("upload_video_file", "Maximum file size 100 MB")
                return render(request, index_template_name, {"form": video_upload_form})

        if sequence_length <= 0:
            video_upload_form.add_error("sequence_length", "Sequence Length must be greater than
0")
            return render(request, index_template_name, {"form": video_upload_form})

        if allowed_video_file(video_file.name) == False:
            video_upload_form.add_error("upload_video_file","Only video files are allowed ")
            return render(request, index_template_name, {"form": video_upload_form})

        saved_video_file = 'uploaded_file_'+str(int(time.time()))+"."+video_file_ext
        if settings.DEBUG:
            with open(os.path.join(settings.PROJECT_DIR, 'uploaded_videos', saved_video_file),
'wb') as vFile:
                shutil.copyfileobj(video_file, vFile)
            request.session['file_name'] = os.path.join(settings.PROJECT_DIR, 'uploaded_videos',
saved_video_file)
        else:
            with open(os.path.join(settings.PROJECT_DIR,
'uploaded_videos','app','uploaded_videos', saved_video_file), 'wb') as vFile:
                shutil.copyfileobj(video_file, vFile)
            request.session['file_name'] = os.path.join(settings.PROJECT_DIR,
'uploaded_videos','app','uploaded_videos', saved_video_file)
        request.session['sequence_length'] = sequence_length
        return redirect('ml_app:predict')
    else:
        return render(request, index_template_name, {"form": video_upload_form})

def predict_page(request):
  if request.method == "GET":
    # Redirect to 'home' if 'file_name' is not in session
    if 'file_name' not in request.session:
```

```
            return redirect("ml_app:home")
        if 'file_name' in request.session:
            video_file = request.session['file_name']
        if 'sequence_length' in request.session:
            sequence_length = request.session['sequence_length']
        path_to_videos = [video_file]
        video_file_name = os.path.basename(video_file)
        video_file_name_only = os.path.splitext(video_file_name)[0]
        # Production environment adjustments
        if not settings.DEBUG:
            production_video_name = os.path.join('/home/app/staticfiles/', video_file_name.split('/')[3])
            print("Production file name", production_video_name)
        else:
            production_video_name = video_file_name

        # Load validation dataset
        video_dataset = validation_dataset(path_to_videos, sequence_length=sequence_length,
transform=train_transforms)

        # Load model
        if(device == "gpu"):
            model = Model(2).cuda()  # Adjust the model instantiation according to your model
structure
        else:
            model = Model(2).cpu()  # Adjust the model instantiation according to your model structure
        model_name = os.path.join(settings.PROJECT_DIR, 'models',
get_accurate_model(sequence_length))
        path_to_model = os.path.join(settings.PROJECT_DIR, model_name)
        model.load_state_dict(torch.load(path_to_model, map_location=torch.device('cpu')))
        model.eval()
        start_time = time.time()
        # Display preprocessing images
        print("<=== | Started Videos Splitting | ===>")
        preprocessed_images = []
```

```
        return redirect("ml_app:home")
        if 'file_name' in request.session:
            video_file = request.session['file_name']
        if 'sequence_length' in request.session:
            sequence_length = request.session['sequence_length']
        path_to_videos = [video_file]
        video_file_name = os.path.basename(video_file)
        video_file_name_only = os.path.splitext(video_file_name)[0]
        # Production environment adjustments
        if not settings.DEBUG:
            production_video_name = os.path.join('/home/app/staticfiles/', video_file_name.split('/')[3])
            print("Production file name", production_video_name)
        else:
            production_video_name = video_file_name

        # Load validation dataset
        video_dataset = validation_dataset(path_to_videos, sequence_length=sequence_length,
transform=train_transforms)

        # Load model
        if(device == "gpu"):
            model = Model(2).cuda()  # Adjust the model instantiation according to your model
structure
        else:
            model = Model(2).cpu()  # Adjust the model instantiation according to your model structure
        model_name = os.path.join(settings.PROJECT_DIR, 'models',
get_accurate_model(sequence_length))
        path_to_model = os.path.join(settings.PROJECT_DIR, model_name)
        model.load_state_dict(torch.load(path_to_model, map_location=torch.device('cpu')))
        model.eval()
        start_time = time.time()
        # Display preprocessing images
        print("<=== | Started Videos Splitting | ===>")
        preprocessed_images = []
```

```
faces_cropped_images = []
    cap = cv2.VideoCapture(video_file)
    frames = []
    while cap.isOpened():
        ret, frame = cap.read()
        if ret:
            frames.append(frame)
        else:
            break
    cap.release()

    print(f"Number of frames: {len(frames)}")
    # Process each frame for preprocessing and face cropping
    padding = 40
    faces_found = 0
    for i in range(sequence_length):
        if i >= len(frames):
            break
        frame = frames[i]

        # Convert BGR to RGB
        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Save preprocessed image
        image_name = f"{video_file_name_only}_preprocessed_{i+1}.png"
        image_path = os.path.join(settings.PROJECT_DIR, 'uploaded_images', image_name)
        img_rgb = pImage.fromarray(rgb_frame, 'RGB')
        img_rgb.save(image_path)
        preprocessed_images.append(image_name)

        # Face detection and cropping
        face_locations = face_recognition.face_locations(rgb_frame)
        if len(face_locations) == 0:
            continue
```

```
    top, right, bottom, left = face_locations[0]
    frame_face = frame[top - padding:bottom + padding, left - padding:right + padding]

    # Convert cropped face image to RGB and save
    rgb_face = cv2.cvtColor(frame_face, cv2.COLOR_BGR2RGB)
    img_face_rgb = pImage.fromarray(rgb_face, 'RGB')
    image_name = f"{video_file_name_only}_cropped_faces_{i+1}.png"
    image_path = os.path.join(settings.PROJECT_DIR, 'uploaded_images', image_name)
    img_face_rgb.save(image_path)
    faces_found += 1
    faces_cropped_images.append(image_name)

print("<=== | Videos Splitting and Face Cropping Done | ===>")
print("--- %s seconds ---" % (time.time() - start_time))

# No face detected
if faces_found == 0:
    return render(request, 'predict_template_name.html', {"no_faces": True})

# Perform prediction
try:
    heatmap_images = []
    output = ""
    confidence = 0.0

    for i in range(len(path_to_videos)):
        print("<=== | Started Prediction | ===>")
        prediction = predict(model, video_dataset[i], './', video_file_name_only)
        confidence = round(prediction[1], 1)
        output = "REAL" if prediction[0] == 1 else "FAKE"
        print("Prediction:", prediction[0], "==", output, "Confidence:", confidence)
        print("<=== | Prediction Done | ===>")
        print("--- %s seconds ---" % (time.time() - start_time))
```

```
            # Uncomment if you want to create heat map images
            # for j in range(sequence_length):
            #     heatmap_images.append(plot_heat_map(j, model, video_dataset[i], './',
video_file_name_only))

        # Render results
        context = {
            'preprocessed_images': preprocessed_images,
            'faces_cropped_images': faces_cropped_images,
            'heatmap_images': heatmap_images,
            'original_video': production_video_name,
            'models_location': os.path.join(settings.PROJECT_DIR, 'models'),
            'output': output,
            'confidence': confidence
        }

        if settings.DEBUG:
            return render(request, predict_template_name, context)
        else:
            return render(request, predict_template_name, context)

    except Exception as e:
        print(f"Exception occurred during prediction: {e}")
        return render(request, 'cuda_full.html')
def about(request):
    return render(request, about_template_name)

def handler404(request,exception):
    return render(request, '404.html', status=404)
def cuda_full(request):
    return render(request, 'cuda_full.html')
```

**styles.css**

```
  body {
   height: 100%;
}
.bg {
   /* The image used */
   background-image: url("/static/images/background1.png");

   -webkit-background-size: cover;
   -moz-background-size: cover;
   -o-background-size: cover;

   /* Center and scale the image nicely */
   background-position: center;
   background-repeat: no-repeat;
   background-size: cover;



}

section{
   padding-top: 60px;
}

.width-300{
   width: 300px;
   margin: auto;
   padding: 20px;
   box-shadow: 0 0px 9px 2px #ccc;
}

.width-400{
   width: 400px;
   margin: auto;
```

```
    padding: 20px;
    margin-top: 80px;
    margin-bottom: 150px;
    box-shadow: 0 0px 9px 2px #ccc;
}

.width-500{
    width: 500px;
    margin: auto;
    padding: 20px;
    box-shadow: 0 0px 9px 2px #ccc;
}

#videos{
    display: none;
}

canvas {
    position: absolute;
    top: 0;
    left: 0;
}

.preprocess {
    padding-right: 20px;
    padding-bottom: 50px;
}

#preprocessed_images {
  white-space: nowrap;
  width: auto;
  height: 250px;
  padding: 20px;
  overflow-x: scroll;
```

```
  overflow-y: hidden;
  box-shadow: 0 0px 9px 2px #ccc;
}

#faces_images {
    white-space: nowrap;
    width: auto;
    height: 150px;
    padding: 20px;
    overflow-x: scroll;
    overflow-y: hidden;
    box-shadow: 0 0px 9px 2px #ccc;
}

.faces {
    padding-right: 20px;
    padding-bottom: 50px;
}

#heatmap_images{
    white-space: nowrap;
    width: auto;
    height: 200px;
    padding: 20px;
    overflow-x: scroll;
    overflow-y: hidden;
    box-shadow: 0 0px 9px 2px #ccc;
    margin-bottom: 20px;
}

.heat-map {
    padding-right: 20px;
    padding-bottom: 50px;
}
```

# 7. TESTING

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and code generation. The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

**7.1 Testing Objectives:**

- To ensure that during operation the system will perform as per specification.
- To make sure that system meets the user requirements during operation
- To make sure that during the operation, incorrect input, processing and output will be detected.
- To see that when correct inputs are fed to the system the outputs are correct
- To verify that the controls incorporated in the same system as intended
- Testing is a process of executing a program with the intent of finding an error
- A good test case is one that has a high probability of finding an as yet undiscovered error

The software developed has been tested successfully using the following testing strategies and any errors that are encountered are corrected and again the part of the program or the procedure or function is put to testing until all the errors are removed. A successful test is one that uncovers an as yet undiscovered error.

Note that the result of the system testing will prove that the system is working correctly. It will give confidence to system designer, users of the system, prevent frustration during implementation process etc.

**7.2 Testing Methodologies:**

- White box testing.
- Black box testing.
- Unit testing.

- Integration testing.
- User acceptance testing.
- Output testing.
- Validation testing.
- System testing.

**1)White Box Testing:**

White box testing is a testing case design method that uses the control structure of the procedure design to derive test cases. All independents path in a module are exercised at least once, all logical decisions are exercised at once, execute all loops at boundaries and within their operational bounds exercise internal data structure to ensure their validity. Here the customer is given three chances to enter a valid choice out of the given menu. After which the control exits the current menu.

**2)Black Box Testing:**

Black Box Testing attempts to find errors in following areas or categories, incorrect or missing functions, interface error, errors in data structures, performance error and initialization and termination error. Here all the input data must match the data type to become a valid entry.

**3)Unit Testing:**

Unit testing focuses verification effort on the smallest unit of Software design that is the module. Unit testing exercises specific paths in a module's control structure to ensure complete coverage and maximum error detection. This test focuses on each module individually, ensuring that it functions properly as a unit. Hence, the naming is Unit Testing.

**4)Integration Testing:**

Integration testing addresses the issues associated with the dual problems of verification and program construction. After the software has been integrated a set of high order tests are conducted. The main objective in this testing process is to take unit tested modules and builds a program structure that has been dictated by design.

The following are the types of Integration Testing:

- **Top Down Integration:**

This method is an incremental approach to the construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main program module.

- **Bottom Up Integration:**

This method begins the construction and testing with the modules at the lowest level in the program structure. Since the modules are integrated from the bottom up, processing required for modules subordinate to a given level is always available and the need for stubs is eliminated.

**5)User acceptance Testing:**

User Acceptance of a system is the key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with the prospective system users at the time of developing and making changes wherever required. The system developed provides a friendly user interface that can easily be understood even by a person who is new to the system.

**6)Output Testing:**

After performing the validation testing, the next step is output testing of the proposed system, since no system could be useful if it does not produce the required output in the specified format. Asking the users about the format required by them tests the outputs generated or displayed by the system under consideration. Hence the output format is considered in 2 ways – one is on screen and another in printed format.

**7)Validation Testing:**

Validation testing is generally performed on the following fields:

- **Text Field:**

The text field can contain only the number of characters lesser than or equal to its size. The text fields are alphanumeric in some tables and alphabetic in other tables. Incorrect entry always flashes and error message.

- **Numeric Field:**

The numeric field can contain only numbers from 0 to 9. An entry of any character flashes an error messages. The individual modules are checked for accuracy and what it has to perform.

- **Preparation of Test Data:**

Taking various kinds of test data does the above testing. Preparation of test data plays a vital role in the system testing. After preparing the test data the system under study is tested using that test data. While testing the system by using test data errors are again uncovered and corrected by using above testing steps and corrections are also noted for future use.

- **Using Live Test Data:**

Live test data are those that are actually extracted from organization files. After a system is partially constructed, programmers or analysts often ask users to key in a set of data from their normal activities. Then, the systems person uses this data as a way to partially test the system. In other instances, programmers or analysts extract a set of live data from the files and have them entered themselves.

- **Using Artificial Test Data:**

Artificial test data are created solely for test purposes, since they can be generated to test all combinations of formats and values. In other words, the artificial data, which can quickly be prepared by a data generating utility program in the information systems department, make possible the testing of all login and control paths through the program.

The most effective test programs use artificial test data generated by persons other than those who wrote the programs. Often, an independent team of testers formulates a testing plan, using the systems specifications.

### 7.3 User Training:

Whenever a new system is developed, user training is required to educate them about the working of the system so that it can be put to efficient use by those for whom the system has been primarily designed. For this purpose the normal working of the project was demonstrated to the prospective users. Its working is easily understandable and since the expected users are people who have good knowledge of computers, the use of this system is very easy.

### 7.4 Maintainance:

This covers a wide range of activities including correcting code and design errors. To reduce the need for maintenance in the long run, we have more accurately defined the user's requirements during the process of system development. Depending on the requirements, this system has been developed to satisfy the needs to the largest possible extent. With development in technology, it may be possible to add many more features based on the requirements in future. The coding and designing is simple and easy to understand which will make maintenance easier.

### 7.5 Testing Strategy:

A strategy for system testing integrates system test cases and design techniques into a well planned series of steps that results in the successful construction of software. The testing strategy must co-operate test planning, test case design, test execution, and the resultant data collection and evaluation

.A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high level tests that validate major system functions against user requirements.
Software testing is a critical element of software quality assurance and represents the ultimate review of specification design and coding.

### 7.5.1 System Testing:

Software once validated must be combined with other system elements (e.g. Hardware, people, database). System testing verifies that all the elements are proper and that overall system function performance is achieved. It also tests to find discrepancies between the system and its original objective, current specifications and system documentation.

### 7.5.2 Unit Testing:

In unit testing different are modules are tested against the specifications produced during the design for the modules. Unit testing is essential for verification of the code produced during the coding phase, and hence the goals to test the internal logic of the modules. Using the detailed design description as a guide, important Conrail paths are tested to uncover errors within the boundary of the modules. This testing is carried out during the programming stage itself. In this type of testing step, each module was found to be working satisfactorily as regards to the expected output from the module. In Due Course, latest technology advancements will be taken into consideration. As part of technical build-up many components of the networking system will be generic in nature so that future projects can either use or interact with this.

# 8. OUTPUT SCREENS

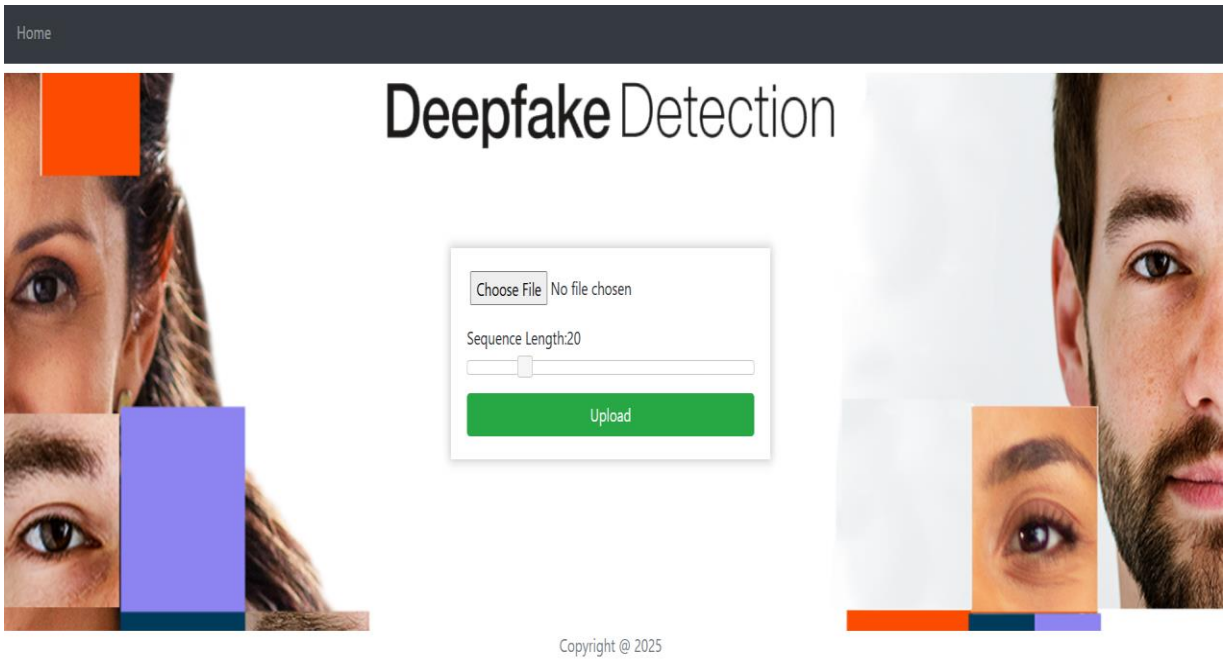



**Fig 8.1 Home Page**

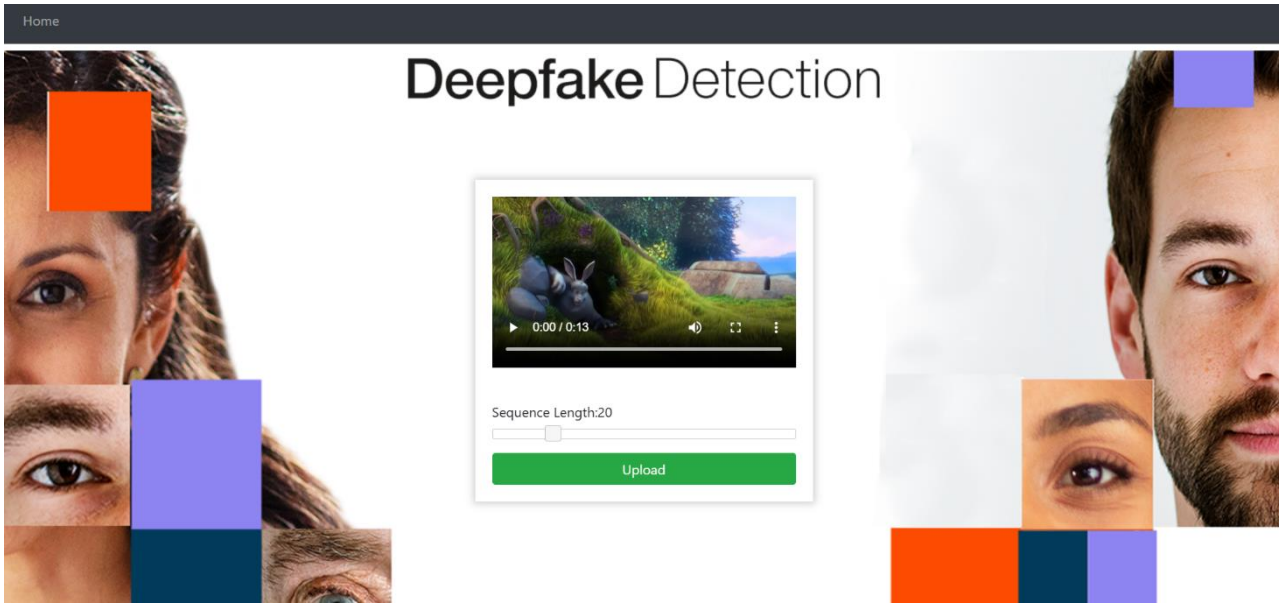



**Fig 8.2 Upload Video**

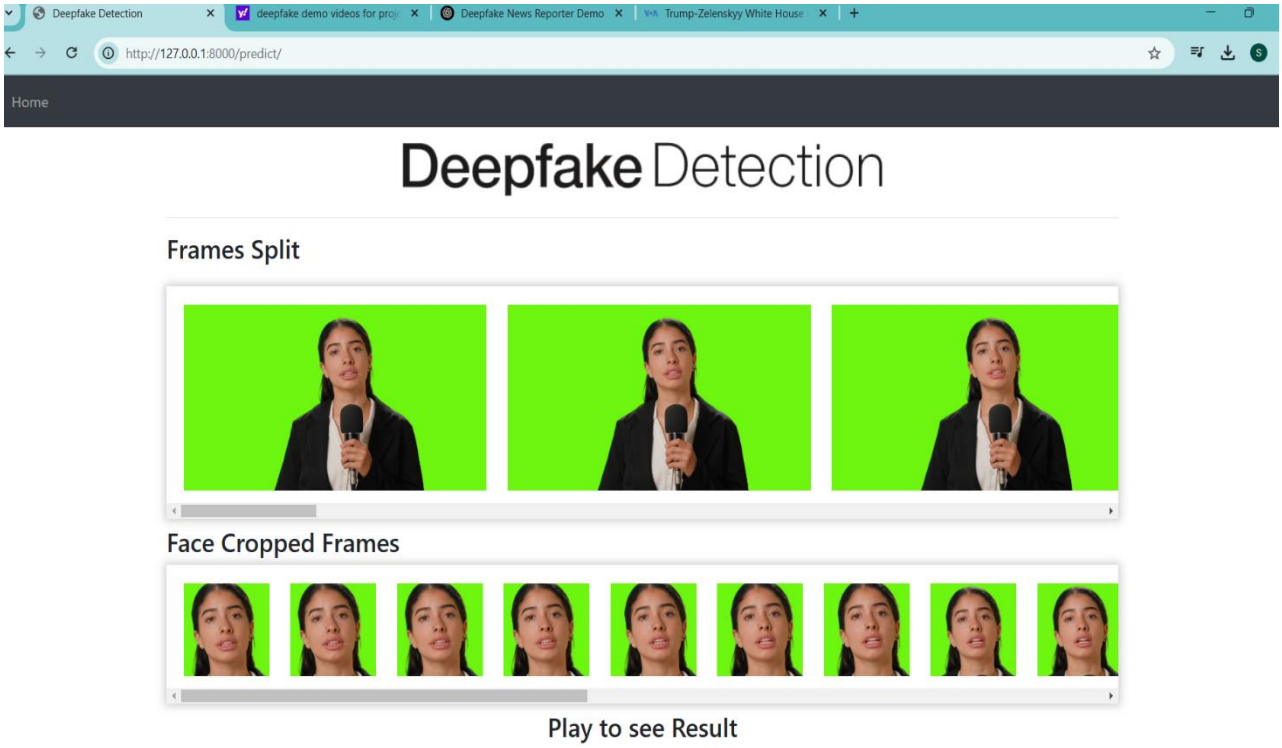



**Fig 8.3: video diving into frame**

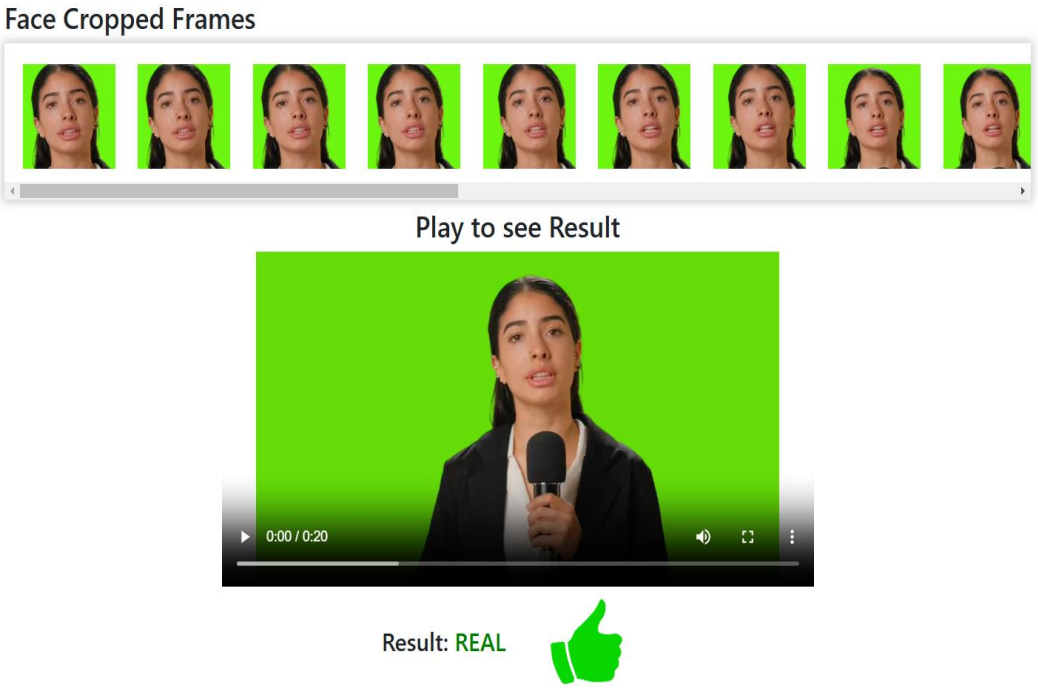



**Fig 8.5 Detecting video is real or fake**

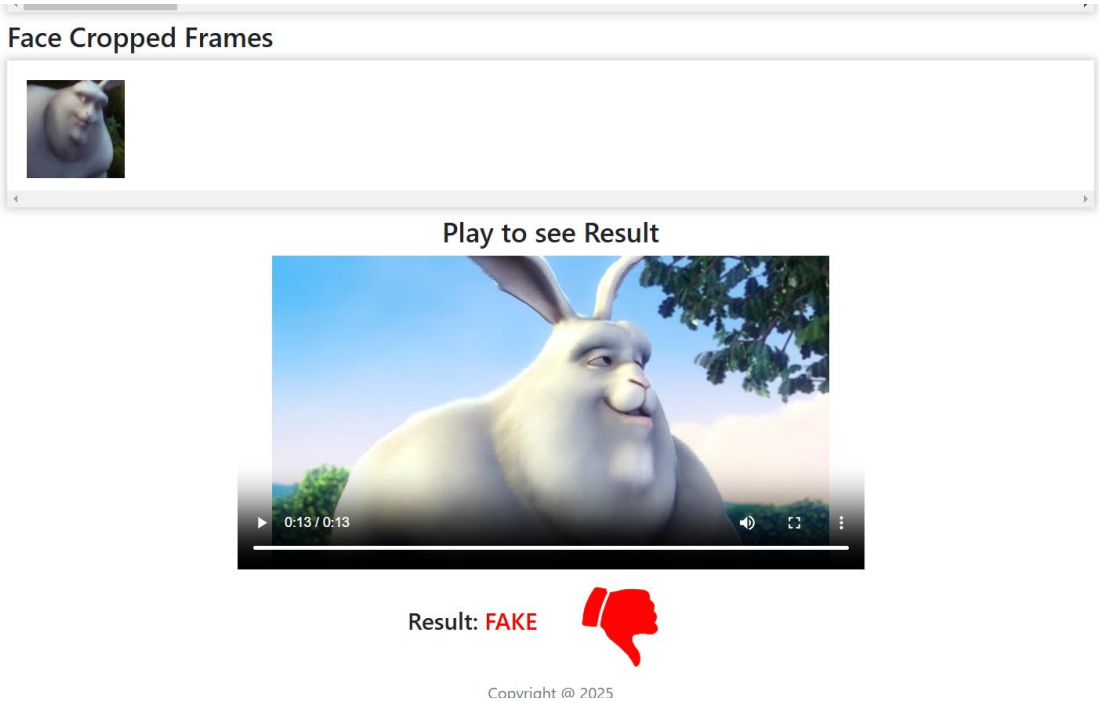



**Fig 8.5 Detecting video is real or fake**

# 9. CONCLUSION AND FUTURE SCOPE

## 9.1 Conclusion

The Deepfake Video Detection project successfully implements an advanced deep learning-based approach to identify manipulated videos with high accuracy. By leveraging CNN, ResNet, LSTM, and Vision Transformer (GenConViT) architectures, the system efficiently extracts spatial and temporal features to detect deepfakes. The combination of frame-based analysis, face detection, and classification models ensures reliable and automated detection. The project's effectiveness is further enhanced by robust preprocessing techniques and dataset management. The results demonstrate a high detection accuracy, making this system a valuable tool for combating misinformation, ensuring media authenticity, and strengthening digital security. Future improvements may focus on enhancing real-time detection, expanding datasets, and optimizing computational efficiency for large-scale deployment.

Deepfake video detection has become a crucial area of research due to the increasing misuse of AI-generated fake videos for misinformation, fraud, and identity manipulation. This project focuses on leveraging deep learning models such as CNN, ResNet, LSTM, and Vision Transformers (GenConViT) to accurately distinguish between real and fake videos. The system processes videos by extracting frames, detecting faces, and analyzing subtle inconsistencies that indicate tampering. Advanced architectures like ResNext for feature extraction and LSTM for temporal analysis improve classification accuracy. The integration of GenConViT further enhances performance by capturing both local and global dependencies in video sequences. By utilizing state-of-the-art deep learning techniques, this project provides a reliable and automated solution for detecting deepfake videos, which can be beneficial in areas such as digital forensics, media verification, and cybersecurity.

The Deepfake Video Detection project addresses the growing challenge of detecting AI-generated fake videos by integrating state-of-the-art deep learning models such as CNN, ResNet, LSTM, and GenConViT (Generative Convolutional Vision Transformer). These models work together to analyze both spatial and temporal features, ensuring a comprehensive evaluation of video authenticity. The system employs preprocessing techniques, including frame extraction, face detection, cropping, and alignment, to enhance feature extraction accuracy. The use of OpenCV, Dlib, and FFmpeg aids in efficient video processing, while deep learning frameworks like TensorFlow and PyTorch power the training and inference process.

## 9.2 Future Scope:

The future scope of deepfake video detection is vast, with numerous advancements and improvements that can enhance the accuracy, efficiency, and applicability of the system.

1. **Real-Time Detection** – Future models can be optimized to analyze and detect deepfake videos in real-time, making them more useful for social media platforms, news agencies, and law enforcement.
2. **Multi-Modal Detection** – Incorporating audio and text analysis along with video detection can improve accuracy, as deepfake videos often include manipulated voices and misleading subtitles.
3. **Enhanced Dataset Diversity** – Expanding datasets to include more realistic and sophisticated deepfakes will improve model robustness and adaptability to new deepfake generation techniques.
4. **Explainable AI (XAI) Integration** – Implementing interpretable AI models will help users understand why a video is classified as fake, increasing trust and reliability in the system.
5. **Lightweight & Edge AI Models** – Optimizing deepfake detection for mobile devices and edge computing will allow for faster and more accessible detection, reducing dependency on high-end GPUs.
6. **Blockchain for Verification** – Storing video authenticity records on a blockchain network can ensure the integrity of digital media and prevent unauthorized tampering.
7. **Adversarial Deepfake Detection** – As deepfake generation methods evolve, adversarial training techniques can help models adapt dynamically to new manipulation techniques, maintaining high detection accuracy.
8. **Integration with Social Media & Law Enforcement** – Collaboration with social media platforms, cybersecurity agencies, and law enforcement can enable automatic detection and flagging of manipulated content to prevent misinformation spread.
9. **Integration with Blockchain and Digital Watermarking** – Implementing blockchain-based verification and digital watermarking can help track the origin of videos and ensure authenticity. This will provide a secure framework for verifying media sources and preventing tampering.
10. **Enhanced AI Models with Explainability** – The use of explainable AI (XAI) techniques will allow deepfake detection models to provide insights into why a video is classified as fake or real, making detection systems more transparent and trustworthy.

11. **Cross-Modal Deepfake Detection** – Future detection systems will integrate multi-modal learning, analyzing not just video frames but also audio, facial expressions, and gestures to improve accuracy and robustness against sophisticated deepfakes.
12. **Lightweight and Edge AI Models** – Developing low-computation deepfake detection models for mobile devices and IoT systems will enable widespread accessibility and on-device verification, reducing dependency on cloud-based infrastructure.
13. **Legal and Ethical Frameworks** – Collaboration between AI researchers, governments, and cybersecurity experts will lead to the development of global regulations and policies that mandate deepfake detection and impose legal actions against malicious usage.
14. **Large-Scale Deployment and Cloud-Based Detection** – With increasing deepfake threats, scalable cloud-based solutions integrated with AI-powered media forensics will allow organizations and individuals to verify the authenticity of videos efficiently.

# 10.BIBLIOGRAPHY

[1] Joshua Brockschmidt, Jiacheng Shang, and Jie Wu. On the Generality of Facial Forgery Detection. In 2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems Workshops (MASSW), pages 43–47. IEEE, 2019.

[2] Yuezun Li, Ming-Ching Chang, and Siwei Lyu. In Ictu Oculi: Exposing AI Generated Fake Face Videos by Detecting Eye Blinking. arXiv preprint arXiv:1806.02877v2, 2018.

[3] TackHyun Jung, SangWon Kim, and KeeCheon Kim. Deep-Vision: Deepfakes Detection Using Human Eye Blinking Pattern. IEEE Access, 8:83144–83154, 2020.

[4] Konstantinos Vougioukas, Stavros Petridis, and Maja Pantic. Realistic Speech-Driven Facial Animation with GANs. International Journal of Computer Vision, 128:1398–1413, 2020.

[5] Hai X. Pham, Yuting Wang, and Vladimir Pavlovic. Generative Adversarial Talking Head: Bringing Portraits to Life with a Weakly Supervised Neural Network. arXiv preprint arXiv:1803.07716, 2018.

[6] Yuezun Li, Siwei Lyu, "ExposingDF Videos By Detecting Face Warping Artifacts," in arXiv:1811.00656v3.

[7] Yuezun Li, Ming-Ching Chang and Siwei Lyu "Exposing AI Created Fake Videos by Detecting Eye Blinking" in arxiv.

[8] Huy H. Nguyen , Junichi Yamagishi, and Isao Echizen " Using capsule networks to detect forged images and videos ".

[9] Umur Aybars Ciftci, ˙Ilke Demir, Lijun Yin "Detection of Synthetic Portrait Videos using Biological Signals" in arXiv:1901.02212v2. 35

[10] https://www.kaggle.com/c/deepfake-detection-challenge/data

[11] Liu, M. Y., Huang, X., Mallya, A., Karras, T., Aila, T., Lehtinen, J., and Kautz, J. (2019). Few-shot unsupervised image-to-image translation. In Proceedings of the IEEE International Conference on Computer Vision (pp. 10551-10560).