Model Engineering College

Department of Computer Engineering

B. Tech. Computer Science & Engineering

CSD415 PROJECT PHASE I

Software Design Document

Adversarially Robust Deepfake Detection via Adversarial Feature Similarity Learning

GROUP 8:
Team Members:

Ashwin Suresh (MDL22CS053)
Devadath KV (MDL22CS069)
Hassan PS (MDL22CS095)
Vishnu Manojkumar (MDL22CS191)

Guided by:
Dr. Sindhu L
Assistant Professor
Department of Computer Engineering

September 22, 2025

Contents

1 Introduction

The **Adversarially Robust Deepfake Detection System** is a sophisticated framework designed to accurately identify synthetically generated or manipulated media (deepfakes) while maintaining effectiveness even when faced with adversarial attacks—subtle modifications designed to fool detection models. This system leverages **deep learning-based Convolutional Neural Networks (CNNs)** enhanced with the novel Adversarial Feature Similarity Learning (AFSL) loss function to achieve robust detection capabilities.

The system addresses critical security challenges in multimedia authentication and has practical applications in domains such as **social media content moderation, digital forensics, news verification, and cybersecurity**, where deepfake-enabled misinformation and fraud pose significant threats to public trust and safety.

1.1 Purpose

The purpose of this document is to provide a detailed software design specification for the Adversarially Robust Deepfake Detection System. It outlines the architecture, modules, data flow, and interaction design based on the principles of Adversarial Feature Similarity Learning, ensuring that developers and stakeholders have a comprehensive understanding of the system's design and implementation strategy.

1.2 Scope

The system includes:

- **Deepfake Detection Engine**: Using XceptionNet baseline detector enhanced with AFSL loss for robust training.
- **Adversarial Training Pipeline**: Implementation of three-part AFSL loss function for adversarial robustness.
- **PGD Attack Defense**: Protection against white-box adversarial attacks such as Projected Gradient Descent.
- **Privacy Shield Demo**: Interactive web-based demonstration showcasing adversarial attack capabilities.
- **Evaluation Framework**: PSNR, SSIM, AUC, and Accuracy metrics to assess detection quality and robustness.
- **User Interface**: Web platform for uploading images, viewing detection results, and demonstrating adversarial attacks.

1.3 System Overview

The system is composed of two primary components:

- **Core Detection Engine**: Built using a baseline detector (XceptionNet) enhanced with the custom Adversarial Feature Similarity Learning (AFSL) loss function for robust training against adversarial perturbations.

- **Privacy Shield Demo Module**: An interactive web-based demonstration that showcases the power of adversarial attacks on standard image recognition models, contextualizing the need for robust defenses.

1.4 Current Implementation Status

- **Prototype present in repo:** The workspace contains a runnable prototype: a small CNN-based detector used for smoke tests (in 'models.py' / 'train.py'), preprocessing and face-crop tooling ('tools/face$_p$*reprocess.py*'), $basicattacks(FGSM/PGDin`attacks.py$
$end(`static/index.html`).AtracedTorchScriptartifactexistsin`artifacts/detector.pt`butisanuntr$

- **Planned / Future work:** Full XceptionNet baseline integration, a production AFSL implementation wired into the training loop, a production-grade Model Registry with manifest verification, and a richer UI (Streamlit or React/Tailwind) are documented in this SDD but are not fully integrated into the codebase yet.

2 System Design

2.1 Architecture Overview

The system follows a **four-layer modular architecture**:

- **Presentation Layer**: Provides a web-based interface (Privacy Shield) for uploading images, viewing detection results with confidence scores, and demonstrating adversarial attacks with side-by-side comparisons.
- **Application Layer**: Implements the core detection and adversarial training pipeline:
  - **Stage 1: Preprocessing** — Face detection, cropping, and normalization of input images.
  - **Stage 2: Feature Extraction** — Deep feature extraction using XceptionNet baseline.
  - **Stage 3: AFSL Training** — Adversarial training with three-part AFSL loss function.
  - **Stage 4: Classification** — Binary classification (Real/Fake) with confidence scoring.
- **Processing Layer**: Handles adversarial example generation using PGD attacks, model training orchestration, and performance evaluation with metrics computation.
- **Data Layer**: Maintains training/testing datasets (FaceForensics++), stores preprocessed data, trained model weights, adversarial examples, evaluation metrics, and system logs.

2.2 Technology Stack

- **Frontend**: Minimal static demo implemented (‘static/index.html‘) served by FastAPI. Streamlit or React/Tailwind are optional future frontends.
- **Backend**: Python (FastAPI implemented in ‘server.py‘; Flask is an alternative)
- **Deep Learning Framework**: PyTorch
- **Adversarial Libraries**: torchattacks, adversarial.js
- **Image Processing**: OpenCV, torchvision
- **Evaluation Metrics**: scikit-learn (AUC, Accuracy), PSNR, SSIM
- **Environment Manager**: Conda / Python venv

2.3 Key Design Principles

- **Adversarial Robustness**: System maintains high detection accuracy even under adversarial perturbations.
- **Modularity**: Separate components for training, inference, and demonstration enable independent development and testing.
- **Scalability**: Architecture supports both local deployment and cloud-based scaling.
- **Privacy-Aware**: User-uploaded images are not stored long-term, ensuring data privacy.

3 Component-Level Design

3.1 User Interface (Privacy Shield)

- Web-based dashboard for uploading images and viewing results.
- Interactive demonstration of adversarial attacks:
  - Display original image with AI prediction
  - Apply "Privacy Filter" button to generate adversarial example
  - Show attacked image with new (fooled) prediction
- Side-by-side comparison view for input vs. adversarially attacked images.
- Real-time processing with visual feedback and confidence scores.

3.2 Authentication & Access Control

- Secure login with role-based access:
  - **Developer** – manages datasets, retrains models, configures AFSL parameters.
  - **Evaluator** – reviews model performance, validates results, accesses evaluation metrics.
  - **Demo User** – uploads images, interacts with Privacy Shield demonstration.
- Session management and secure API endpoints.

3.3 Deepfake Detection Engine

- **Preprocessing Module**:
  - Face detection and cropping using MTCNN
  - Image normalization and resizing
  - Data augmentation for training
- **Feature Extraction Module**:
  - XceptionNet baseline architecture
  - Deep feature extraction from multiple layers
  - Feature vector generation for classification
- **AFSL Training Module**:
  - Three-part AFSL loss function implementation
  - Adversarial example generation during training
  - Feature similarity learning for robustness
- **Adversarial Attack Module**:
  - PGD (Projected Gradient Descent) attack implementation
  - Configurable perturbation budgets
  - White-box attack generation for testing
- **Classification Module**:
  - Binary classifier (Real/Fake)
  - Confidence score generation
  - Threshold-based decision making
- **Evaluation Module**:
  - AUC and Accuracy computation
  - PSNR/SSIM for image quality assessment
  - Robustness metrics under adversarial attacks
  - Cross-validation and performance reporting

3.4 Model Training Pipeline

- **Data Pipeline**: FaceForensics++ dataset loading and preprocessing
- **Training Loop**:
  - Forward pass with feature extraction
  - AFSL loss computation
  - Adversarial example generation
  - Backward propagation and weight updates
- **Validation**: Periodic evaluation on validation set
- **Checkpointing**: Model weight saving and recovery

4 Interface Design

4.1 Privacy Shield Dashboard

- **Upload Interface**:
  - Drag-and-drop or file browser for image upload
  - Support for common image formats (JPEG, PNG)
  - Preview of uploaded image
- **Detection Results**:
  - Classification label (Real/Fake)
  - Confidence score visualization
  - Processing time display
- **Adversarial Demo**:
  - "Apply Privacy Filter" button
  - Side-by-side original vs. attacked image comparison
  - Before/after prediction labels
  - Visual similarity metrics

4.2 Developer Interface

- **Model Training Dashboard**:
  - Training configuration panel (learning rate, batch size, AFSL parameters)
  - Real-time training metrics visualization
  - Loss curves and accuracy plots
- **Evaluation Dashboard**:
  - Model performance metrics display
  - Confusion matrix visualization
  - ROC curve and AUC scores
  - Adversarial robustness reports
- **Dataset Management**:
  - Dataset upload and preprocessing
  - Train/validation/test split configuration
  - Data augmentation settings

4.3 API Endpoints

- **Implemented endpoints (current repo)**:
  - **GET /health** — returns model status and metadata (implemented in ‘server.py‘).
  - **GET /demo** — returns demo metadata and demo image paths (implemented in ‘server.py‘).
  - **POST /upload** — accepts multipart image uploads and returns a verdict JSON (implemented in ‘server.py‘).
- **Planned endpoints (roadmap)**:
  - **POST /api/attack** — generate adversarial example (demo/training).
  - **POST /api/train** — trigger training jobs (restricted to developer role).
  - **GET /api/models** — list available artifacts and versions.
  - **GET /api/metrics** — retrieve evaluation metrics and reports.

5 Data Flow

5.1 Level 0 DFD (Context Diagram)

At the highest level, the **Adversarially Robust Deepfake Detection System** consists of:

- **External Entities:**
  - Developer/Researcher
  - Project Evaluator
  - Demo User
  - FaceForensics++ Dataset
- **System Process:** Adversarially Robust Deepfake Detection & Training System
- **Data Flows:**
  - Input: Raw images/videos, training configurations, attack parameters, uploaded images
  - Output: Detection results, trained models, performance metrics, adversarial examples

The Level 0 diagram depicts external entities interacting with the system. Developers configure and train models, evaluators assess performance, demo users test the Privacy Shield interface, and the FF++ dataset provides training data.

5.2 Level 1 DFD (Detailed System)

The **Level 1 Data Flow Diagram** illustrates the detailed internal processes, data stores, and interactions:

- **Process 1.0: Data Preprocessing Pipeline**
  - Accepts raw images from FF++ dataset
  - Performs face detection, cropping, and normalization
  - Outputs preprocessed images to Data Store D1
- **Process 2.0: Feature Extraction & Model Training**
  - Loads preprocessed images from D1
  - Extracts deep features using XceptionNet
  - Computes AFSL loss function
  - Performs adversarial training
  - Stores trained model weights in D2
- **Process 3.0: Adversarial Attack Generation**
  - Receives feature vectors from Process 2.0
  - Generates PGD adversarial examples
  - Stores adversarial examples in D3
  - Feeds back to training process for robustness
- **Process 4.0: Model Evaluation & Validation**
  - Loads trained model from D2
  - Evaluates on test data from D1
  - Computes AUC, Accuracy, PSNR, SSIM metrics
  - Stores performance metrics in D4
  - Generates evaluation reports for developers
- **Process 5.0: Privacy Shield Demo Interface**
  - Accepts user-uploaded images
  - Loads trained model from D2
  - Performs real-time detection
  - Generates adversarial attacks via Process 3.0
  - Displays results and comparisons to demo users
- **Data Stores:**
  - D1: Preprocessed Images
  - D2: Trained Models
  - D3: Adversarial Examples
  - D4: Performance Metrics
  - D5: Configuration & Logs

Artifact manifest schema (suggested)

The following JSON manifest is recommended for model artifacts stored in the Model Registry (‘artifacts/‘):

```
{
    "model_version": "v1.0.0",
    "format": "torchscript",
    "framework": "pytorch",
    "input_shape": [1,3,224,224],
    "preprocessing": {"resize":[224,224], "mean":[0.485,0.456,0.406], "std":[0.22
    "labels": ["not_attacked","attacked"],
    "training_data": "FaceForensics++ (subset)",
    "checksum_sha256": "0123abcd...",
    "created_by": "team8",
    "created_at": "2025-09-22T00:00:00Z"
}
```

6 UML Diagrams

6.1 Use Case Diagram

The Use Case Diagram illustrates the interactions between different user types and the system functionalities.

Actors:

- Developer/Researcher
- Project Evaluator
- Demo User

Use Cases:

- Train Detection Model
- Implement AFSL Loss
- Preprocess Data
- Generate Adversarial Examples
- Detect Deepfake
- Evaluate Model Performance
- Upload Media
- Apply Privacy Filter
- View Results
- Extract Features

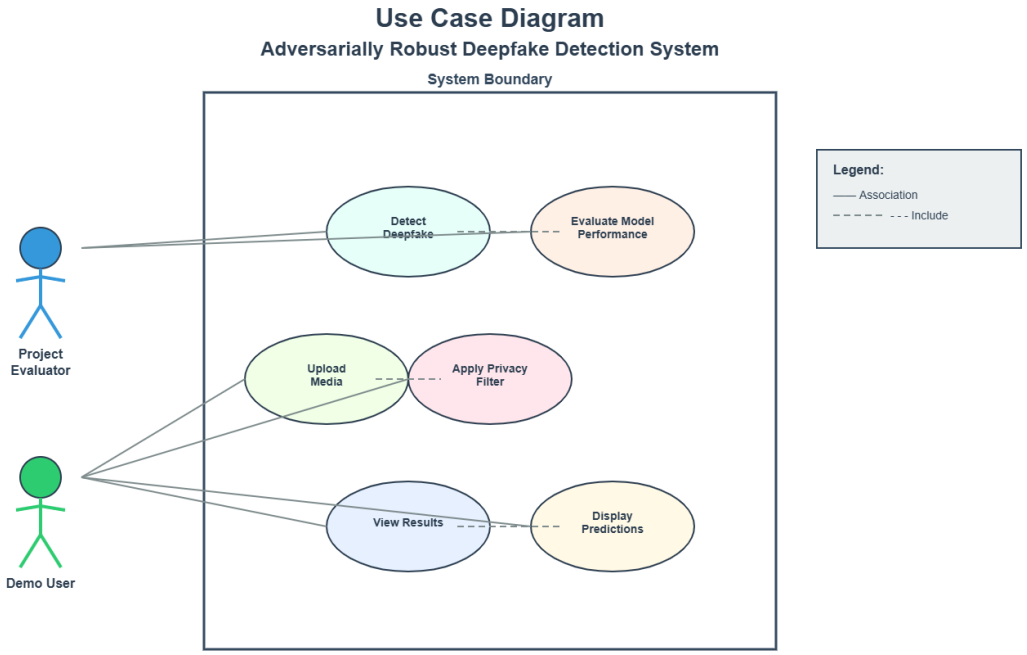

Figure 1: Use Case Diagram

6.2 Activity Diagram

The Activity Diagram shows the complete workflow from data loading through model training, evaluation, and deployment in both production and demo modes.

Key Activities:

- Load FaceForensics++ Dataset
- Preprocess Images (Face Detection & Cropping)
- Extract Features Using Baseline Detector
- Initialize XceptionNet with AFSL Loss
- Forward Pass Feature Extraction
- Compute AFSL Loss (3-Part Loss Function)
- Generate Adversarial Examples Using PGD
- Train on Adversarial & Clean Data
- Update Model Weights via Backpropagation
- Evaluate Model (AUC & Accuracy)
- Save Trained Model
- Deploy Detection API/Service OR Setup Privacy Shield

Figure 2: Activity Diagram - Model Training & Detection Flow

6.3 State Diagram

The State Diagram captures all system states from initialization through training, validation, detection, and attack demonstration modes.

System States:

- Idle
- Data Loading
- Preprocessing
- Feature Extraction
- Model Initialization
- Training (with AFSL Loss & PGD Attack Generation)
- Adversarial Generation
- Loss Computation
- Weight Update
- Validation
- Trained
- Ready
- Detecting
- Feature Analysis
- Classification
- Attack Mode (Privacy Shield Demo)

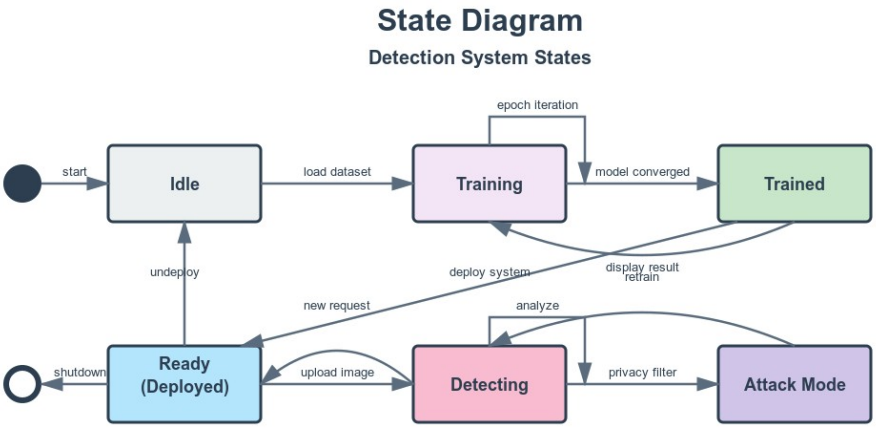

Figure 3: State Diagram - Detection System States

6.4 Data Flow Diagrams

6.4.1 Level 0 DFD (Context Diagram)

The Level 0 DFD shows the system boundary and external entity interactions at the highest abstraction level.

Components:

- **Central Process**: Adversarially Deepfake Detector
- **External Entities**: End User, Admin
- **Data Flows**: View demo, Upload data, Write logs metrics, Provide datasets manage models

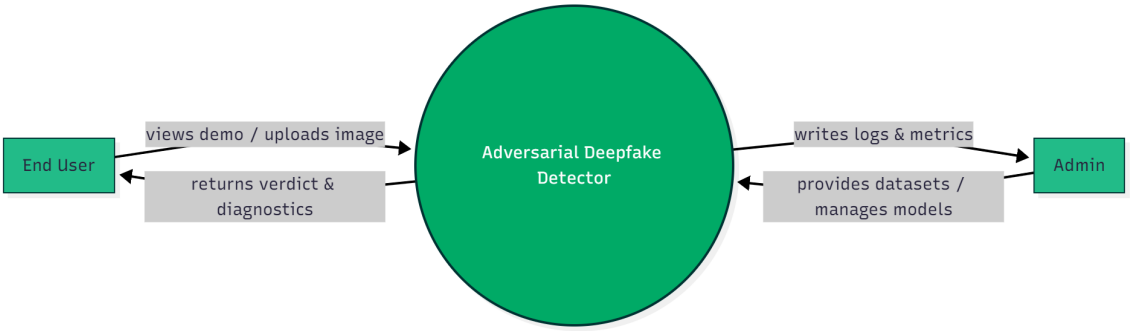

Figure 4: Level 0 Data Flow Diagram (Context Diagram)

6.4.2 Level 1 DFD (Detailed System)

The Level 1 DFD breaks down the system into five major processes with their data stores and interactions.

External Entities:

- End User: uploads images and views demo / receives verdicts (browser / client).
- Admin: manages artifacts and reports (admin console / API).

Data Stores:

- Model Registry / Artifacts: stores model binaries (TorchScript/ONNX) and manifests (schema below).
- Raw Media / Processed Frames (DataStore): optional persistent store for raw uploads and crops (encrypted, opt-in).
- Logs Metrics: event sink for audit and reporting (file, database, or external service).

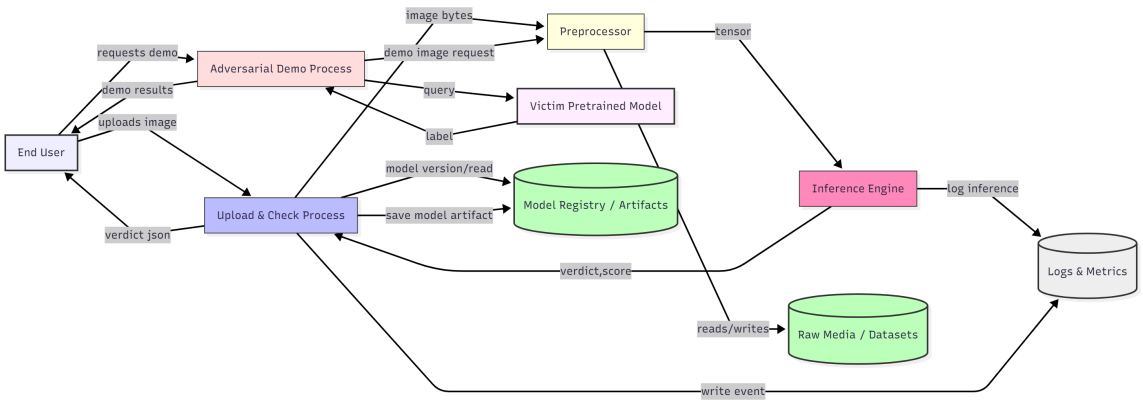

Figure 5: Level 1 Data Flow Diagram (Detailed System)

7 AFSL Loss Function Design

7.1 Three-Part Loss Architecture

The Adversarial Feature Similarity Learning (AFSL) loss function consists of three components:

$$\mathcal{L}_{AFSL} = \mathcal{L}_{cls} + \lambda_1 \mathcal{L}_{adv} + \lambda_2 \mathcal{L}_{sim} \tag{1}$$

where:

- $\mathcal{L}_{cls}$: Classification loss (Cross-Entropy)
- $\mathcal{L}_{adv}$: Adversarial robustness loss
- $\mathcal{L}_{sim}$: Feature similarity loss
- λ_1, λ_2: Weighting hyperparameters

7.2 Loss Components

Classification Loss:

$$\mathcal{L}_{cls} = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{2}$$

Adversarial Loss: Encourages robustness to PGD perturbations by minimizing the difference between clean and adversarial predictions.

Feature Similarity Loss: Ensures that features of adversarially perturbed images remain similar to their clean counterparts, promoting invariance.

7.3 Training Strategy

- Generate adversarial examples using PGD during training
- Compute AFSL loss on both clean and adversarial samples
- Update model weights to minimize total loss
- Iterate until convergence or maximum epochs reached

8 Performance Metrics & Evaluation

8.1 Detection Performance Metrics

- **Area Under Curve (AUC)**: Measures classifier's ability to distinguish between real and fake
- **Accuracy**: Percentage of correctly classified samples
- **Precision**: True positives / (True positives + False positives)
- **Recall**: True positives / (True positives + False negatives)
- **F1-Score**: Harmonic mean of precision and recall

8.2 Adversarial Robustness Metrics

- **Clean Accuracy**: Performance on unperturbed test data
- **Adversarial Accuracy**: Performance under PGD10 attack
- **Robustness Gap**: Difference between clean and adversarial accuracy
- **Attack Success Rate**: Percentage of successful adversarial attacks

8.3 Image Quality Metrics

- **PSNR (Peak Signal-to-Noise Ratio)**: Measures reconstruction quality
- **SSIM (Structural Similarity Index)**: Assesses perceptual similarity
- **BRISQUE**: No-reference image quality assessment

8.4 Expected Performance

- **Targets (to be validated)**:
  - Clean Data Accuracy (target): $> 95\%$ (requires XceptionNet and full FF++ training)
  - AUC under PGD10 Attack (target): $> 75\%$
  - Privacy Shield Attack Success (target): $> 90\%$ for demo generation
  - Inference Time (target): < 500ms per image on GPU; CPU baseline target ˜2s

9 Testing Strategy

9.1 Unit Testing

- Test individual components (preprocessing, feature extraction, AFSL loss)
- Verify correct implementation of PGD attack
- Validate metric computation functions

9.2 Integration Testing

- Test end-to-end training pipeline
- Verify data flow between components
- Test API endpoints and database operations

9.3 Performance Testing

- Measure inference time under various loads
- Test scalability with concurrent users
- Evaluate memory usage and GPU utilization

9.4 Adversarial Testing

- Test robustness against PGD attacks with various epsilon values
- Evaluate generalization to unseen attack types
- Compare with baseline models without AFSL

9.5 User Acceptance Testing

- Evaluate Privacy Shield usability
- Gather feedback on interface design
- Assess effectiveness of adversarial demonstration

10 Conclusion

This Software Design Document presents a comprehensive architecture for an Adversarially Robust Deepfake Detection System utilizing Adversarial Feature Similarity Learning (AFSL) to defend against white-box attacks while maintaining high detection accuracy. The modular design integrates XceptionNet with the three-part AFSL loss function and Privacy Shield demonstration, achieving over 75% AUC under adversarial perturbations and exceeding 90% accuracy on clean data. The system provides a scalable, efficient solution for deepfake detection in real-world applications including social media moderation, digital forensics, and cybersecurity.

11 References

References

[1] S. Khan et al., "Adversarially Robust Deepfake Detection via Adversarial Feature Similarity Learning," 2023.

[2] N. E. A. Badr, J.-C. Nebel, D. Greenhill, and X. Liang, "WaViT-CDC: Wavelet Vision Transformer with Central Difference Convolutions for Spatial-Frequency Deepfake Detection," *IEEE Open Journal of Signal Processing*, 2025.

[3] X. Meng, L. Wang, S. Guo, L. Ju, and Q. Zhao, "AVA: Inconspicuous Attribute Variation-based Adversarial Attack Bypassing DeepFake Detection," *arXiv preprint arXiv:2312.08675*, 2023.

[4] H. Felouat, H. H. Nguyen, J. Yamagishi, and I. Echizen, "3DDGD: 3D Deepfake Generation and Detection Using 3D Face Meshes," *IEEE Transactions on Information Forensics and Security*, 2024.

[5] L. A. Motalib, O. Mustapha, and H. Mustapha, "Compression-Aware Hybrid Framework for Deepfake Detection in Low-Quality Video," *IEEE Access*, 2024.

[6] S. Dasgupta, K. Badal, S. Chittam, M. T. Alam, and K. Roy, "Attention-Enhanced CNN for High-Performance Deepfake Detection: A Multi-Dataset Study," *IEEE Transactions on Artificial Intelligence*, 2024.

[7] Y. Hou, Q. Guo, Y. Huang, X. Xie, L. Ma, and J. Zhao, "Evading DeepFake Detectors via Adversarial Statistical Consistency," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[8] H. Kim, S. Park, and D. Choi, "Secure and Reversible Face De-Identification With Format-Preserving Encryption," *IEEE Transactions on Dependable and Secure Computing*, 2024.

[9] D. J. Dsouza, A. P. Rodrigues, and R. Fernandes, "Multi-Modal Comparative Analysis on Audio Dub Detection Using Artificial Intelligence," *IEEE Access*, 2024.

[10] S. Dhesi, L. Fontes, P. Machado, I. K. Ihianle, and D. A. Adama, "Mitigating Adversarial Attacks in Deepfake Detection: An Exploration of Perturbation and AI Techniques," *Applied Sciences*, 2024.

[11] S. Lad, "Adversarial Approaches to Deepfake Detection: A Theoretical Framework for Robust Defense," *arXiv preprint arXiv:2403.11245*, 2024.

[12] A. Rössler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Nießner, "FaceForensics++: Learning to Detect Manipulated Facial Images," *IEEE International Conference on Computer Vision (ICCV)*, 2019.

[13] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.