ENHANCING DEEPFAKE DETECTION GENERALIZATION THROUGH COMPONENT-BASED DEVELOPMENT IN A WEB PLATFORM

KOH YEONG KEONG

UNIVERSITI TUNKU ABDUL RAHMAN

ENHANCING DEEPFAKE DETECTION GENERALIZATION THROUGH COMPONENT-BASED DEVELOPMENT IN A WEB PLATFORM

KOH YEONG KEONG

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Software Engineering with Honours

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

October 2025

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Name		Koh Yeong Keong
ID No.	:	2105656
Date	:	29/04/2025

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"ENHANCING DEEPFAKE DETECTION GENERALIZATION THROUGH COMPONENT-BASED DEVELOPMENT IN A WEB PLATFORM"** was prepared by **Koh Yeong Keong** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Software Engineering with Honours at Universiti Tunku Abdul Rahman

Approved by,

Signature : 

Supervisor : KELWIN TAN SEEN TIONG

Date : 17/10/2025

COPYRIGHT STATEMENT

This final year project report is submitted in partial fulfilment of the requirements for the degree of Software Engineer at Universiti Tunku Abdul Rahman (UTAR). This final year project report represents the work of the author, except where due acknowledgement has been made in the text. No part of this final year project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ABSTRACT

This report outlines the design, development, and evaluation of a deepfake detection system aimed at providing an accessible and scalable solution for detecting manipulated media. The system leverages advanced machine learning models, including both single-model and ensemble-based detection methods, to identify deepfakes in images. The platform supports easy image uploads, efficient model processing, and reliable result presentation, offering users the ability to choose between various detection models based on their needs.

Key features include user authentication and role management, image validation, preprocessing, and real-time inference with confidence scores. The system utilizes a modular architecture to integrate new models seamlessly, ensuring scalability and maintainability. Performance benchmarks are met, including a processing time of less than 800ms per image and a 99.9% uptime for system reliability. The accuracy of the ensemble detection method is validated through extensive testing on benchmark datasets, achieving a high F1-score.

This project addresses the growing concern of deepfake threats in digital media and aims to provide an easy-to-use, robust tool for both non-technical users and advanced administrators. The system is designed with a focus on usability, accuracy, performance, and security, ensuring it meets the challenges posed by modern deepfake detection.

Keywords: Machine Learning, Generative AI, Deepfake, Component-based, Image Classification, Artificial Intelligence, AI Generalization

Subject Area: QA75.5-76.95 Computer Science

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 General Introduction

Deepfake technology, a term derived from "deep learning" and "fake," refers to synthetic media generated using artificial intelligence (AI) to manipulate or fabricate images, videos, audio, or text with striking realism (Anna, 2024). This technology relies on advanced deep learning techniques, particularly Generative Adversarial Networks (GANs), where a generator creates fake content and a discriminator evaluates its authenticity, iteratively refining outputs until they appear genuine (Yasar, Barney, and Wigmore, 2024).

Deepfakes first gained prominence in 2017 when a Reddit user demonstrated face-swapping in videos, sparking debates about their ethical implications (Simonite, 2019). Since then, advancements in AI have made the technology more sophisticated and accessible. By 2023, over 500,000 deepfake videos circulated online, spanning applications from entertainment to disinformation campaigns (Jacobson, 2024). Tools like Deepswap and FaceApp now enable even novices to create convincing synthetic media, amplifying both innovation and misuse (*12 Best Deepfake Sites & Apps in 2025 [FREE included]*, no date).

While deepfakes offer niche benefits—such as de-aging actors in films, enhancing medical training simulations, or creating immersive educational content—their rapid evolution has raised urgent ethical and societal concerns (Greggwirth, 2023).

1.2 Importance of the Study

The proliferation of deepfakes poses significant risks to misinformation, fraud, personal privacy, security, and even national security. For example, a fabricated video of Ukrainian President Volodymyr Zelenskyy falsely urging surrender during the Russia-Ukraine war demonstrated how deepfakes can destabilize trust in institutions and democratic processes (Ebaker, 2023). Similarly, deepfakes enable sophisticated financial fraud, such as the 2020 \$35 million scam where criminals used synthetic audio to impersonate a corporate executive (Business Today, 2025). Privacy violations, particularly non-consensual

deepfake pornography targeting women, inflict lasting emotional and reputational harm, while altered medical imagery risks misdiagnosis and insurance fraud.

Existing detection systems struggle to keep pace with the rapid advancement of deepfake creation tools. Classifiers trained on specific datasets or GAN architectures often fail to generalize across diverse manipulation methods, leading to outdated and ineffective solutions (Ramanaharan, Guruge, and Agbinya, 2025). This inadequacy underscores the urgent need for adaptable, modular detection systems capable of addressing evolving synthetic media.

The societal implications are profound: unchecked deepfakes could undermine public trust in media, destabilize democratic processes, and erode personal reputations (Gorbel, no date). Robust detection solutions are essential to mitigate these risks, protect individuals, and preserve the integrity of information in an increasingly digital world (Anna, 2024; Greggwirth, 2023). Addressing the challenge of generalization in detection methods is critical to ensuring reliable performance across real-world scenarios, ultimately safeguarding privacy, security, and trust in digital media.

1.3 Problem Statement

1.3.1 PS1: Generalization of Deepfake Detection Models

Deepfake detection models face a major challenge in generalization, which refers to a model's ability to apply its learned knowledge to new and unseen deepfake manipulations. Currently, most deepfake detection systems are trained on a specific set of datasets and manipulation types, leading to limitations in their ability to detect novel forms of deepfake content. For instance, detection models trained on face-swapping deepfakes often fail when exposed to new deepfake techniques, such as style transfer-based deepfakes. The problem of generalization is illustrated in *Utility of Deep Learning Features for Facial Attributes Manipulation Detection* 2020 which shows the detection accuracy of models when tested on different and all manipulation types revealing a significant performance drop across all detection methods. From the result of the study, it have shown the average detection accuracy under novel manipulations is much lower compared to models tested on previously seen

manipulations, underscoring the need for models capable of generalizing across different types of deepfakes (Afchar et al., 2018; Li et al., 2020).

1.3.2 PS2: Lack of Diverse Datasets for Training Deepfake Detection Models

The lack of diverse datasets is a key limitation in the development of effective deepfake detection systems. Most existing datasets, such as FaceForensics++ and Celeb-DF, focus primarily on face-swapping manipulations and may not represent newer deepfake techniques. The lack of diversity means that detection systems trained on these limited datasets often fail when confronted with new forms of manipulation, resulting in low accuracy and high error rates when tested on unseen manipulation types. *Utility of Deep Learning Features for Facial Attributes Manipulation Detection 2020* highlights the detection accuracy of models when tested with novel manipulation type data, revealing that the performance can vary significantly based on the type of manipulation and dataset used for training. This demonstrates that detection accuracy is much lower when models are tested on diverse types of manipulations, underlining the need for a broader range of datasets to train more robust, adaptable models (Afchar et al., 2018; Gandhi et al., 2021).

1.3.3 PS3: Accessibility of Deepfake Detection Tools (Web Application)

Another critical issue is the inaccessibility of deepfake detection tools, which are often complex and require specialized technical knowledge to operate. Most current detection systems are designed for use by experts in machine learning or computer vision, which limits their usability for the general public. To address this challenge, deepfake detection tools should be accessible via a web-based application that allows users to upload media and receive results quickly and easily. This would democratize the ability to detect deepfakes, making it available to individuals, organizations, and institutions that need to verify media authenticity but lack the technical expertise. A user-friendly web application would allow anyone with an internet connection to check whether a piece of media has been manipulated, ensuring that deepfake detection becomes an accessible tool for all, not just experts (Gandhi et al., 2021).

1.4 Aim and Objectives

1.4.1 O1: To Develop a Generalizable Deepfake Detection System

This objective addresses the problem of generalization by developing a detection system capable of adapting to a wide range of deepfake techniques, regardless of the manipulation type or the dataset used to create it. The system will be trained using diverse datasets that cover various deepfake generation methods, such as audio manipulation, face aging, and style transfer deepfakes, ensuring that it can handle different types of manipulations without a significant drop in accuracy. The model will also be designed to maintain high detection performance even when exposed to new, previously unseen deepfake methods, thus addressing the core issue of generalization in deepfake detection (Afchar et al., 2018; Li et al., 2020).

1.4.2 O2: To Collect and Integrate a Diverse Dataset for Training Deepfake Detection Models

- Problem Addressed: PS2: Lack of Diverse Datasets for Training Deepfake Detection Models

This objective seeks to overcome this limitation by compiling a comprehensive and diverse dataset that covers multiple deepfake generation techniques and manipulation types. The dataset will include deepfakes generated using various GAN architectures, autoencoders, and style transfer methods. It will also incorporate diverse manipulation techniques, such as smile alteration, gender switching, and aging effects, to ensure that the detection system is exposed to a wide range of synthetic media. By training the detection model on this diverse dataset, it will become more robust and adaptable, improving its ability to detect a broader array of deepfake manipulations and making the detection system more effective in real-world scenarios (Gandhi et al., 2021; Afchar et al., 2018).

1.4.3 O3: To Develop an Accessible Web-Based Deepfake Detection Tool

- Problem Addressed: PS3: Accessibility of Deepfake Detection Tools (Web Application)

This objective aims to democratize access to deepfake detection tools, making them available to a much wider audience, including journalists, media organizations, educators, and the general public. The application will be designed for ease of use, with no need for advanced knowledge of machine learning or AI algorithms. The goal is to create a web-based platform that allows anyone with an internet connection to verify the authenticity of media in real time, helping to combat the spread of misinformation, fraud, and defamation caused by manipulated content. By providing an easy-to-use detection tool, this objective seeks to make deepfake verification accessible to all (Gandhi et al., 2021).

1.5 Proposed Solution

To address the critical challenges of generalization, adaptability, and practical deployment in deepfake detection, this project proposes a component-based framework designed to enhance detection robustness across diverse datasets and evolving synthetic media techniques. The solution integrates three core innovations, aligned with the project's objectives and methodology.

First, the Component-Based Modular Architecture employs a modular design to decouple detection components, enabling independent training and testing of models tailored to specific datasets or architectures. Each component comprises three units: a Computation Unit that handles model-specific computations (e.g., GAN artifact detection, frequency analysis) while maintaining fixed behavior to ensure consistency; an Output Unit that aggregates results from individual components, providing a unified detection outcome; and a Connector Unit that facilitates seamless communication between components, allowing incremental integration of new models (e.g., diffusion models) without disrupting existing workflows.

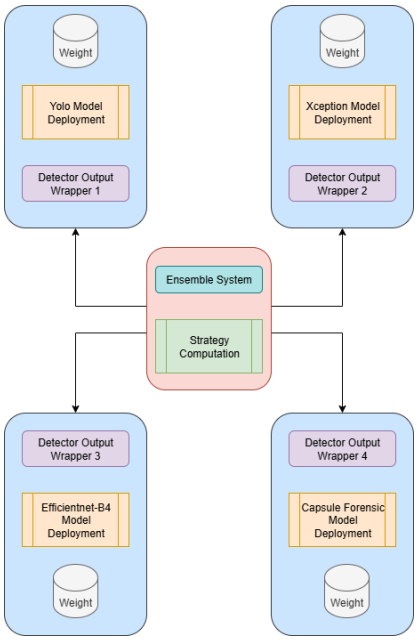

Figure 1: Component-based Architecture as Solution

Second, to mitigate dataset bias and improve generalization, the Diverse Dataset Generation and Integration component curates a comprehensive dataset spanning multiple generative models, including Unconditional GANs like StyleGAN, proGAN, SAGAN, and BigGAN for high-fidelity facial synthesis; Conditional GANs like CycleGAN (face-swapping), StarGAN (attribute editing), and Face2Face (expression manipulation); Auto-Encoders such as FaceForensics++ benchmark images generated via the faceswap tool; and Perceptual Loss Models like Cascaded Refinement Networks (CRN) and Implicit Maximum Likelihood Estimation (IMLE).

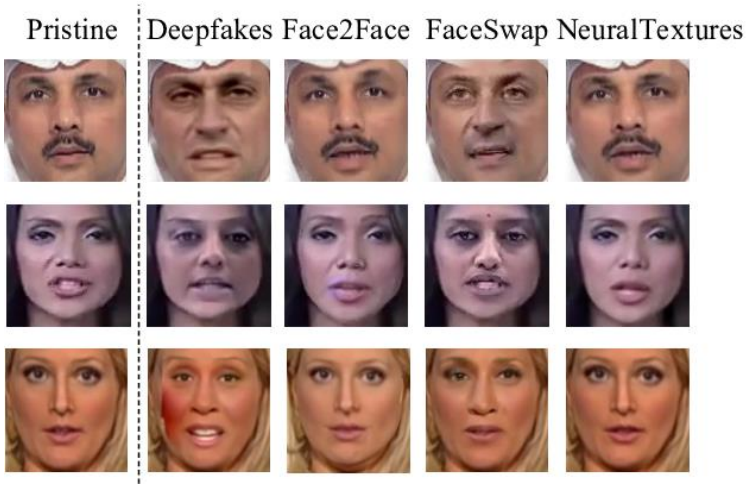

Figure 2: Referencing Image for Deepfake Generated By Differnt GAN Generator (Xu, Raja and Pedersen, 2022)

Third, a Practical Web Application for Real-World Deployment is developed to democratize access to robust deepfake detection. Key features include Multi-Model Support, allowing users to select detection components optimized for specific manipulation types (e.g., face-swaps, expression edits); Adaptive Updates, which allow new components (e.g., for diffusion models) to be added without requiring full system retraining; and Cross-Domain Validation, using metrics like average precision and adaptation rate to ensure performance across unseen datasets. In implementation, Stage 4 (Application Development) leverages the component-based architecture to ensure modularity and scalability, and Stage 5 (Evaluation) tests the application's accuracy on novel manipulation types (e.g., lip-sync forgeries) and computational efficiency.

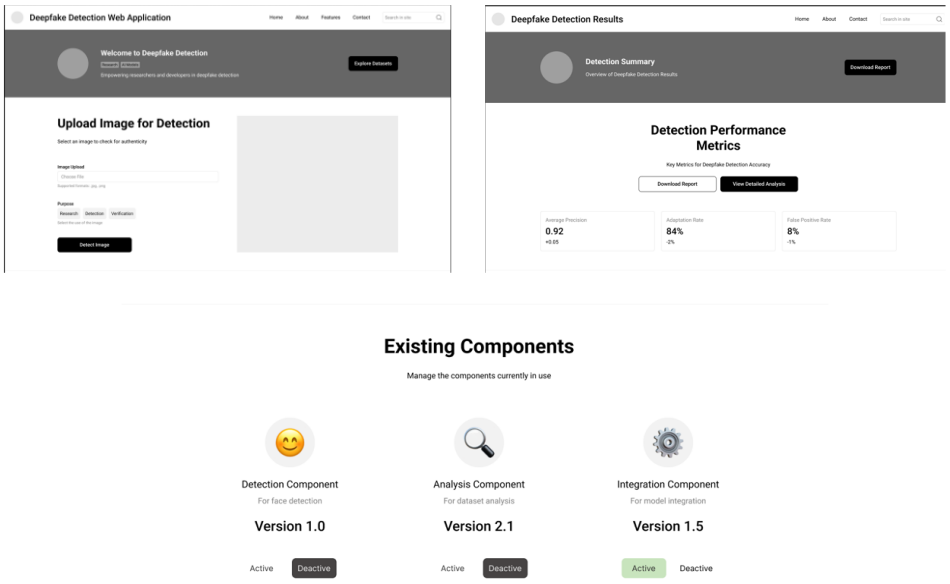

Figure 3: Sample interface design for Proposed web application

1.6 Scope and Limitation of the Study

1.6.1 Scope:

1.6.1.1 Features and Modules to be Developed

Deepfake Detection Application: The core functionality of the system, allowing users to upload images that will be classified as either "real" or "fake" using AI-powered models like Convolutional Neural Networks (CNNs) and the ensemble detection method.

Ensemble Detection System: The system will incorporate multiple models using ensemble strategies like majority voting, weighted averaging, and confidence-based selection. This ensemble method will improve accuracy and decision-making reliability.

Model Management: The system will allow administrators to manage models, including adding, updating, and deactivating them through configuration files (YAML/JSON). The platform will also support retraining with new datasets, ensuring adaptability to emerging deepfake technologies.

Dataset Management: The platform will provide access to curated datasets containing both real and deepfake images, which users can explore, download for research purposes, or use for training new models.

1.6.1.2 Intended Users or Target Audience

Researchers and Academics: They will use the platform to analyze deepfake detection techniques and experiment with different models and datasets.

Developers and AI Engineers: These users will leverage the platform for integrating new models, experimenting with datasets, and optimizing detection performance.

General Public and Media: The public can upload images for detection, providing an easy way to verify the authenticity of images they encounter online.

Institutions (Universities, Law Enforcement): These users will use the platform for verifying documents, images, and media in academic, legal, and security contexts.

1.6.1.3 Technologies or Platforms to be Used

Frontend: The web application will use modern web technologies such as HTML, CSS, and JavaScript, leveraging frameworks like React.js or Vue.js for creating responsive and interactive interfaces.

Backend: Python will be used for developing AI and machine learning models, leveraging deep learning frameworks like TensorFlow and PyTorch. Server-side logic will be managed using Flask for API interactions.

Database: PhpMyAdmin will be used to store user data, metadata, and model performance metrics.

Containerization: Docker will be employed for containerization and orchestration, ensuring smooth deployment and management of the system components.

1.6.1.4 Types of Data or Inputs the System Will Handle

User-uploaded Images: The main input, which can be either real or deepfake images.

Model Data: Pre-trained models deployment script and model weight used for deepfake detection.

Metadata: Information about each processed image, including the prediction (real or fake), confidence scores, and model performance metrics

1.6.2 Limitation:

1.6.2.1 Dataset Limitations

The system relies on datasets for training its detection models. However, datasets may not cover all possible deepfake manipulation types or cross-domain variations such as changes in lighting, resolution, or camera angles. While datasets will include a variety of deepfake techniques, new manipulation methods emerge frequently, and it is impossible to predict and incorporate every potential deepfake generation method.

1.6.2.2 Computational Complexity

Deepfake detection models, particularly those based on deep learning, require substantial computational resources for both training and inference. This may necessitate GPUs or cloud infrastructure to process large datasets and perform real-time detection. Users without access to high-performance hardware could face challenges in training the models, and running real-time inference on large video files could cause latency issues. Cloud computing services, such as AWS and Google Cloud, provide necessary computational resources but may not be accessible to all users, especially those with limited financial resources.

1.6.2.3 Real-Time Detection Limitations

While the system aims for real-time detection, there are inherent limitations. High-resolution images or videos with multiple frames require more processing power and time, potentially leading to delays in providing feedback. As deepfake techniques become more sophisticated, the detection algorithms may need more time to identify subtle manipulation artifacts. Achieving instantaneous results for all media types may not always be feasible, especially for large video files or complex image manipulations.

1.6.2.4 Focus on Specific Manipulation Types

While the system aims for real-time detection, there are inherent limitations. High-resolution images or videos with multiple frames require more processing power and time, potentially leading to delays in providing feedback. As deepfake techniques become more sophisticated, the detection algorithms may need more time to identify subtle manipulation artifacts. Achieving

instantaneous results for all media types may not always be feasible, especially for large video files or complex image manipulations.

CHAPTER 2

LITERATURE REVIEW

2.1 Literature Review of Deepfake Detection

2.1.1 Introduction

This literature review examines the evolution of deepfake generation techniques and the corresponding advancements in detection methodologies, focusing on the technical challenges of generalization, dataset diversity, and real-time applicability. The review also evaluates commercial and open-source detection tools, such as Microsoft Video Authenticator and DeepFake-O-Meter, assessing their strengths in specific manipulation types and weaknesses in scalability or accessibility. By synthesizing these insights, this section establishes the foundation for the proposed modular detection system, which aims to overcome gaps in generalization, usability, and ethical deployment identified in current literature.

2.1.2 Detection Models

DeepfakeBench, introduced in *DeepfakeBench: A Comprehensive Benchmark of Deepfake Detection* by Zhiyuan Yan et al, is a pioneering framework designed to address critical challenges in evaluating deepfake detection models. Traditional approaches often suffer from inconsistent datasets, non-uniform evaluation protocols, and fragmented implementation pipelines, leading to unreliable comparisons between models. In the project, we will use result from this paper, to objectively assess model robustness across different scenarios.

2.1.2.1 Model Training and Assumptions:

Models evaluated within DeepfakeBench are trained on a variety of deepfake datasets, including the newly introduced DF40 dataset, which features 40 distinct deepfake techniques. This diversity ensures that models are exposed to a wide range of manipulation methods during training. The training process assumes that models can learn generalized features capable of detecting various deepfake techniques. However, the effectiveness of these models can vary depending on the specific characteristics of each dataset.

2.1.2.2 Domain issue in Deepfake detection:

Deepfake detection models face a fundamental challenge: balancing performance on familiar data (within-domain) with adaptability to unseen manipulations (cross-domain). This dichotomy reflects the real-world tension between specialization and generalization, where models must excel in controlled settings while remaining robust against evolving threats. In considering such sequence, DeepfakeBench employs both evaluation strategies to assess these competing demands (Yan et al., 2023b):

- **Within-Domain Evaluation:** In this approach, models are trained and tested on data from the same source or domain. This scenario simulates real-world applications where models encounter data similar to their training data. High performance in within-domain evaluations indicates that a model can effectively detect deepfakes within a specific context or dataset.
- **Cross-Domain Evaluation:** Here, models are trained on data from one domain and tested on data from a different, unseen domain. This evaluation simulates real-world scenarios where models must generalize to new, unseen data distributions. High performance in cross-domain evaluations demonstrates a model's robustness and ability to adapt to various deepfake generation techniques and data characteristics.

2.1.2.3 Model Type

In deepfake detection, models are typically categorized into three types based on their feature extraction approaches: naive, spatial, and frequency detectors. Each type has distinct methodologies that influence their accuracy and performance.

1. Naive Detectors:

Naive detectors employ standard convolutional neural networks (CNNs) to perform binary classification between real and fake content without incorporating specialized modules for artifact detection. They rely on the CNN's

ability to learn hierarchical features from the data. While naive detectors can achieve commendable accuracy within the domain they are trained on, their performance may degrade when applied to unseen data or different deepfake generation techniques due to their limited focus on generalized artifact detection. (Yan et al., 2024)

2. Spatial Detectors:

Spatial detectors focus on analyzing pixel-level artifacts and inconsistencies within images or video frames. They often incorporate attention mechanisms or specialized modules to detect anomalies such as unnatural edges, texture inconsistencies, or irregular facial features. By emphasizing these spatial irregularities, these detectors can effectively identify manipulations that are visually apparent. However, their performance can be affected by variations in image quality, resolution, and the presence of compression artifacts, which may obscure the subtle cues they rely on. (NGUYEN et al., 2018)

3. Frequency Detectors:

Frequency detectors analyze images in the frequency domain, targeting spectral anomalies introduced during the deepfake generation process. Techniques such as Discrete Cosine Transform (DCT) or Fast Fourier Transform (FFT) are employed to convert spatial data into frequency components, allowing these models to detect unnatural frequency patterns or compression artifacts. This approach enables frequency detectors to identify subtle manipulations that may not be evident in the spatial domain. Nonetheless, their effectiveness can be influenced by factors like the specific frequency artifacts present in different deepfake techniques and the potential for overfitting to these artifacts, which may limit their generalizability across diverse datasets. (Yan et al., 2024)

Performance Evaluation:

To quantify the trade-offs between model architectures and their real-world applicability, we evaluate five representative detectors from the result of DeepfakeBench's standardized framework. These models—UCF (spatial), SPSL (frequency), Xception (naive), FFD (spatial), and EfficientB4 (naive)—

were selected for their highest performance among the 34 detectors involved. The following comparison highlights critical performance disparities, particularly in cross-domain generalization, while contextualizing their computational demands and specialization biases. The performance result is summarized in the Table 1 below:

Table 1: Performance Comparison of Top-5 Models from DeepFake Benchmark (Yan et al., 2023b)

Detector	**Type**	**Backbone**	**Within-Domain Avg.**	**Cross-Domain Avg.**
UCF	Spatial	Xception	0.9527	0.7801
SPSL	Frequency	Xception	0.9408	0.7875
Xception	Naive	Xception	0.945	0.7718
FFD	Spatial	Xception	0.9434	0.7733
EfficientB4	Naive	Efficient	0.9389	0.7718

2.1.2.4 Discussion on Top-5 Models

UCF (Spatial, Xception Backbone):

UCF is a spatial-based deepfake detector built upon the Xception architecture, enhanced with specialized modules for artifact detection. It utilizes spatial attention mechanisms to focus on localized tampering traces, such as unnatural facial boundaries or inconsistent lighting. (NGUYEN et al., 2018). The Xception backbone, known for its depthwise separable convolutions, efficiently captures hierarchical features while reducing computational overhead. In evaluations, UCF achieved a within-domain average Area Under the Curve (AUC) of 0.9527 and a cross-domain average AUC of 0.7801, ranking in the top three across 11 datasets. However, its high computational load may limit real-world applicability, especially in resource-constrained environments. (Yan et al., 2023a)

SPSL (Frequency, Xception Backbone):

SPSL approaches deepfake detection from a frequency domain perspective, integrating spectral analysis with the Xception architecture. (Liu et al., 2021). By processing images in the frequency domain, SPSL identifies spectral distortions indicative of manipulation, such as unnatural high-frequency patterns. (Liu et al., 2021) This hybrid approach enhances the model's robustness against cross-domain challenges. SPSL achieved the highest cross-domain average AUC of 0.7875 among the evaluated models, though its within-domain performance (AUC of 0.9408) is slightly lower. This trade-off suggests a specialization in detecting diverse deepfake techniques at the expense of some performance on domain-specific data. (Liu et al., 2021)

Xception (Naive, Xception Backbone):

The Xception (Naive) model employs the standard Xception architecture without specialized deepfake detection modules. It relies on hierarchical feature learning to distinguish between real and fake content. (Li et al., 2017) This simplicity contributes to its balanced performance, achieving a within-domain average AUC of 0.945 and a cross-domain average AUC of 0.7718. While it serves as a strong baseline, the lack of specialized mechanisms may limit its effectiveness against sophisticated deepfake generation techniques. (Li et al., 2017)

FFD (Spatial, Xception Backbone):

FFD is tailored for facial forgery detection, integrating facial landmark alignment and local artifact detectors within the Xception framework. By focusing on facial regions prone to manipulation, FFD excels in identifying subtle inconsistencies. (Li et al., 2017) It achieved a within-domain average AUC of 0.9434 and a cross-domain average AUC of 0.7733. However, its specialization may lead to overfitting to facial features, potentially reducing its adaptability to non-facial manipulations. (Li et al., 2017)

5. EfficientB4 (Naive, EfficientNet Backbone):

EfficientB4 utilizes the EfficientNet-B4 architecture, emphasizing resource efficiency and scalability. It incorporates Mobile Inverted Bottleneck Convolutions (MBConv) and Squeeze-and-Excitation blocks to balance model

size and performance. (Potrimba, 2023). With a within-domain average AUC of 0.9389 and a cross-domain average AUC of 0.7718, EfficientB4 offers a lightweight alternative suitable for deployment in resource-constrained settings. However, its peak accuracy is lower compared to more complex models, indicating a trade-off between efficiency and detection performance.

2.1.3 DeepFake Detection Tools:

Deepware Scanner

Deepware Scanner is a specialized tool designed to detect deepfake videos, focusing exclusively on facial manipulations such as face-swaps. It utilizes a convolutional neural network (EfficientNet-B7) pre-trained on ImageNet, fine-tuned using Facebook's DeepFake Detection Challenge (DFDC) dataset to enhance its detection capabilities, optimized for identifying inconsistencies in facial regions. Additionally, the tool employs face clustering techniques to enhance the consistency and reliability of its detection results. In controlled settings, Deepware Scanner has demonstrated a high accuracy rate of ~95–98% when tested on the FaceForensics Actors Dataset. It also maintains robust performance with broader datasets like CFDF, despite unspecified accuracy rate. The tool is capable of processing videos up to 10 minutes in length, but performance may decline with lower-resolution videos. (Hook35, 2021) Deepware Scanner is currently in its beta stage and is available in web-platform, API, and SDK formats.

Figure 4: Logo of Deepware (Deepware, 2025)

Intel’s FakeCatcher

Intel’s FakeCatcher is a real-time deepfake detection tool that utilizes photoplethysmography (PPG) to analyze subtle color changes in facial pixels caused by blood flow, a feature absent in synthetic faces. (Intel, 2022) It also examines eye movements for consistency, aiding in the identification of deepfakes. Regarding the performance, Intel claim that FakeCatcher achieves a 96% accuracy rate. Despite this, it excels in detecting lip-sync deepfakes, such as those generated by MIT. In BBC’s independent test, FakeCatcher correctly identified all MIT-generated lip-sync deepfakes but flagged some real videos as fake due to pixelation or poor lighting. However, the system struggles with pixelated videos, does not analyze audio, and is prone to false positives, flagging real videos as fake. Regarding pricing, specific details are not readily available. Intel has not publicly disclosed the cost of FakeCatcher.

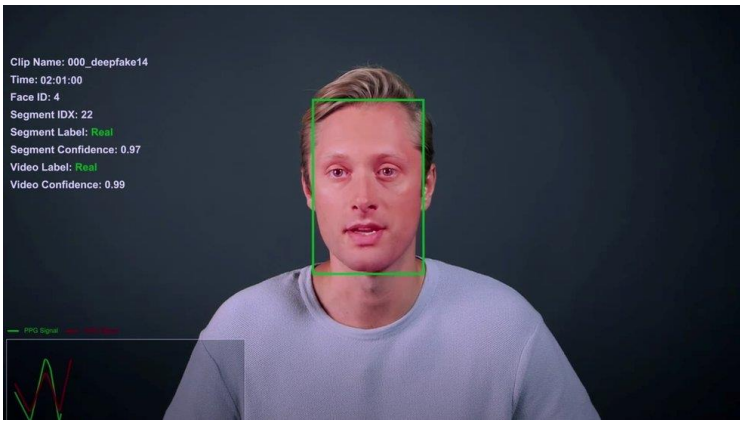

Figure 5: Sample Output from Intel FakeCatcher (Clayton, 2023)

Microsoft Video Authenticator

Microsoft's Video Authenticator is a tool developed to detect deepfake content in photos and videos by analyzing visual cues such as blending boundaries and subtle fading, providing real-time confidence scores for each frame. (Kelion, 2020) While it has demonstrated high accuracy with curated datasets like FaceForensics++ but the specific accuracy is not publicly disclosed. Initially, Video Authenticator was available through the AI Foundation’s Reality Defender 2020 (RD2020) initiative, targeting organizations involved in the democratic process, including news outlets and political campaigns. (Burt, 2020) There is no public information regarding its availability to individual users or details about its pricing.

Figure 6: Sample Output from Microsoft Video Authenticator (Burt, 2020)

Sensity

Sensity is a specialized platform designed to detect deepfakes by analyzing pixel-level inconsistencies, audio patterns, and file structure anomalies. It employs advanced deep learning techniques, including convolutional neural networks (CNNs) and generative adversarial network (GAN) analysis, to identify synthetic content across videos, images, and audio. The platform boasts a 98.7% accuracy rate in detecting AI-generated media, effectively identifying malicious deepfakes on social media and the dark web. (Linkedin.com, 2024) This capability is crucial for sectors like law enforcement and human rights, where combating fraud and non-consensual explicit content is a priority. (Sensity, 2024) The platform serves multiple sectors, including law enforcement, Know Your Customer (KYC) vendors, social media platforms, defense agencies, and digital forensics firms. It is utilized by various organizations, such as TrueMedia, Tradelink, Psyber Labs, Mobbex, 3D Bilism, GlossAI, Confiant, FlipFlop, ArmisonTech, and TransGuard. Sensity offers custom plans tailored to organizational needs, with costs determined upon request. (Sensity, 2023)

Figure 7: Logo of Sensity Ai (Sensity, n.d.)

Reality Defender

Reality Defender is a deepfake detection platform that utilizes advanced AI techniques, including convolutional neural networks (CNNs) and transformer-based architectures, to identify AI-generated content across audio, video, images, and text. The platform employs multiple concurrent models for each media type, enhancing its ability to detect a wide range of generative AI techniques and adapt to emerging threats. In real-world applications, Reality Defender has proven effective in sectors such as contact centers, brand monitoring, real-time video identity authentication, text detection, image authentication, content moderation, and combating disinformation. It has established collaborations with several prominent organizations, including Accenture, Deloitte, IBM, Microsoft, and Nvidia, to enhance deepfake detection capabilities across various industries. Additionally, partnerships with companies like ElevenLabs and TaskUs aim to bolster AI safety and assisted government officials in Canada in identifying and debunking deepfake videos, preserving public trust and integrity. These collaborations underscore Reality Defender's commitment to providing comprehensive solutions for combating AI-generated fraud and disinformation. Regarding cost, Reality Defender offers flexible pricing plans tailored to the specific needs and requirements of each organization.

Figure 8: Logo of Realidy Defender Ai (Realitydefender.com, 2024)

Sentinel AI

Sentinel AI is a deepfake detection platform that utilizes convolutional neural networks (CNNs) and generative adversarial networks (GANs) to identify facial swaps, lip-sync mismatches, and audio manipulations. (Romain, 2023) By

processing video, audio, and metadata, it enhances detection robustness. In controlled environments, Sentinel AI achieves a good accuracy rate (not specific % disclosed) on datasets like FaceForensics++. However, its performance may decrease with high-fidelity deepfakes encountered in real-world scenarios. The platform is employed by governments and organizations such as the European Union External Action and Accelerate Estonia to combat political disinformation and enhancing digital security across various sectors. (Sentinel, n.d) Regarding pricing, specific details are not publicly available. Sentinel AI offers customized solutions tailored to the needs of larger organizations.

Figure 9: Logo of Sentinel AI with slogan (Romain Berg, 2023)

Truepic

Truepic is a platform that ensures the authenticity of digital media through blockchain-based standards like C2PA, embedding tamper-evident certificates into images and videos from creation to publication. This approach allows for verification of media integrity, making it particularly effective in combating synthetic identity fraud and verifying medical scans. Truepic has partnered with various organizations, including Equifax, Davies Group, Northteq, SmartFrame Technologies, and Recall Results, to enhance digital media authenticity, streamline inspections, and innovate product recall processes across multiple industries. However, Truepic's reliance on embedded credentials means it may struggle with high-fidelity deepfakes lacking such metadata. Pricing for Truepic Vision starts at $1,000 per user per month, with costs scaling based on the volume of inspections.

Figure 10: Truepic (Truepic.com, 2022)

iProov's Liveness Detection

iProov is a biometric authentication platform that utilizes its patented Flashmark technology to detect subtle lighting changes and facial movements, ensuring the presence of a live user during authentication. This method effectively differentiates between live individuals and static or synthetic images. iProov boasts a >98% detection rate in controlled environments, excelling in preventing replay attacks and synthetic identity fraud, thereby offering real-time fraud prevention. iProov has formed strategic partnerships with organizations such as Cybernetica, TrustCloud, Biometrid, Authsignal, Matter-ID, and iCloud Identity Inc. to enhance digital identity verification across sectors like government, financial services, and healthcare with notable deployments by the UK Home Office and the U.S. Department of Homeland Security for identity verification purposes. iProov offers a range of service packages and each tailored to organizations with varying user volumes and security requirements. Pricing details, including implementation fees and annual committed fees, are outlined in iProov's official G-Cloud 13 Pricing Document. (G-Cloud 13 Service Definition: iProov Face Verifier, 2022)

Figure 11: iProov (iProov, 2024)

Resemble AI's Detect

Resemble AI's Detect is an advanced neural model designed to identify synthetic audio, video, and images by analyzing subtle inconsistencies across various media types. Its robust detection capabilities make it suitable for real-time applications, such as monitoring live streams for fraudulent activities. To achieve optimal performance, detect requires high-quality audio inputs. The tool is utilized in enterprise security to flag synthetic voices in fraud attempts and in media to verify the authenticity of celebrity voice clones. (Resemble AI, 2024) Besides, Resemble AI has partnered with Carahsoft to provide voice AI and deepfake detection solutions to government agencies while offers a deepfake detection integration for Zoom, analyzing audio snippets during calls to identify potential deepfakes. Regarding the price, Resemble AI offers a range of subscription plans to cater to diverse user needs, including the Starter Plan at $5 per month, the Creator Plan at $19 per month, the Professional Plan at $99 per month, the Scale Plan at $299 per month, the Business Plan at $699 per month, and the Enterprise Plan, which requires direct consultation for pricing. (Resemble AI, 2023)

Figure 12: Resemble AI (Resemble AI, 2024)

DuckDuckGoose

DuckDuckGoose is an AI-driven platform specializing in detecting deepfakes across images, videos, and audio by employing explainable AI (XAI) to highlight manipulated regions, enhancing transparency in detection decisions. It achieves a 99% accuracy rate in identifying deepfakes in images and videos (retained by Phocus) with 0.01% false-rejection rate. The platform is utilized in sectors such as banking to prevent fraud, in legal settings to verify evidence authenticity, and for Know Your Customer (KYC) processes during user onboarding. To be detail, DuckDuckGoose has partnered with Banco Daycoval, bunq and Dutch House by integrating its DeepDetector solution. For the pricing, DuckDuckGoose did not disclose the price publicly (DuckDuckGoose, n.d.)

Figure 13: DuckDuckGoose (Sukrit, 2025)

HyperVerge:

HyperVerge's deepfake detection capabilities are built on a robust technical foundation. The platform utilizes ISO 30107-3 certified AI models for passive liveness detection, distinguishing live users from deepfakes, masks, and screen replays by analyzing micro-movements, such as eye blinks and blood flow patterns, along with texture inconsistencies in real-time. In terms of accuracy and performance, HyperVerge boasts an impressive 98.5% accuracy rate in detecting synthetic media, including deepfakes and GAN-generated faces, on datasets like FaceForensics++. It maintains a low False Acceptance Rate (FAR) of less than 0.1% for spoof attacks, including high-quality deepfakes, and processes liveness checks in under 3 seconds, making it highly effective for real-time fraud prevention during user onboarding. HyperVerge is deployed for SIM card fraud prevention, it has helped telecom companies like Reliance Jio reduce SIM fraud by 99%. In financial services, the platform blocks synthetic identities in loan applications by flagging AI-generated selfies or manipulated documents.

It has also partnered with governments to verify voter identities and counter deepfake-driven disinformation campaigns. Regarding the price, HyperVerge provides deepfake detection services with pricing starting at $0.04 per verification for document quality checks, selfies, and ID forgery, and $0.07 for video KYC verifications.

Figure 14: Hyperverge (Praveen, 2024)

Comparison:

The comparison table provides an overview of included deepfake detection tools, comparing their accuracy, strengths, weaknesses, deployment scenarios, and pricing. It highlights that most tools specialize in different media types (images, videos, audio) and use various approaches, such as AI models, blockchain technology, and facial biomet

Table 2: Overview Comparison of DeepFake Detection Tools

Tool Name	Deployment	Detection Focus	Accuracy	Strengths	Real-World Deployment	Pricing
Deepware Scanner	Single-model	Facial manipulations (face-swaps)	~95-98% (FaceForensics)	Specialized for facial inconsistencies	Beta stage, web-platform, API, SDK available	Free
Intel's FakeCatcher	Single-model	Facial features (blood flow, eye movements)	96%	Real-time detection, excels in lip-sync deepfakes	Deployed for lip-sync detection, BBC test results	Not publicly disclosed
Microsoft Video Authenticator	Single-model	Blending boundaries, subtle fading	Not disclosed	High accuracy with curated datasets	Primarily targeting democratic process, political campaigns	Not publicly disclosed
Sensity	Multi-model	Pixel inconsistencies, audio patterns, file structure anomalies	98.70%	Effective for social media and dark web deepfakes	Deployed in law enforcement, KYC, defense agencies	Custom pricing plans
Reality Defender	Multi-model	Audio, video, images, text	Not disclosed	Multiple models for varied media types, wide industry adoption	Deployed in contact centers, brand monitoring, government	Custom pricing plans
Sentinel AI	Multi-model	Facial swaps, lip-sync, audio	Not disclosed	Robust for political disinformation, customized solutions	Used by EU External Action, Accelerate Estonia	Custom pricing plans
Truepic	Single-model	Blockchain-based media integrity	Not applicable	Effective in synthetic identity fraud and medical scans	Deployed for media authenticity verification in various sectors	$1,000+ per user per month
iProov	Multi-model	Biometric authentication (liveness detection)	>98%	Excellent for preventing fraud in real-time, live user detection	UK Home Office, U.S. Department of Homeland Security	Custom pricing plans
Resemble AI	Multi-model	Audio, video, and images	Not disclosed	Real-time detection, suitable for live stream fraud detection	Deployed in enterprise security, Zoom integration	Plans range from $5 to $699 per month
DuckDuckGoose	Multi-model	Images, videos, audio	95-99%	High accuracy, explainable AI (XAI) transparency	Banco Daycoval, bunq, Dutch House for KYC	Pricing not disclosed
HyperVerge	Multi-model	Deepfakes, liveness, document verification	98.50%	Quick detection (<3s), low FAR (<0.1%), ISO 30107-3 certified	Reliance Jio (SIM fraud reduction), Flip (financial services)	Starts at $0.04 per verification, free trial available

2.1.4 Open-source Tools/Framework:

DeepSafe (Sah, 2023):

https://github.com/siddharthksah/DeepSafe

DeepSafe is an open-source, Streamlit-based web application developed to facilitate deepfake detection research by providing a unified platform for testing and comparing various detection models. It supports multiple pre-trained models, including CNNs and GAN-based detectors, and allows users to integrate custom models. Users can analyze media through direct uploads or by processing URLs, with the platform converting files to standardized formats to ensure compatibility across detectors. DeepSafe offers optional GPU acceleration for faster inference. While DeepSafe includes a benchmarking feature to evaluate models on datasets like FaceForensics++ and Celeb-DF, providing metrics such as accuracy, precision, recall, and inference time, specific accuracy metrics for the included models are not disclosed. DeepSafe serves as a valuable tool for researchers, educators, and developers aiming to analyze and combat deepfake content. (Sah, 2023)

Deepstar (zerofox-oss, 2019):

https://github.com/zerofox-oss/deepstar

Deepstar is an open-source toolkit developed by ZeroFox to aid in the detection, analysis, and mitigation of deepfakes. It offers a suite of tools designed to streamline operations related to deepfake detection research. In technical basis, Deepstar provides code for automating the creation of deepfake datasets, testing, and enhancing detection algorithms. It includes a curated library of deepfake and real videos sourced from platforms like YouTube, facilitating the development and evaluation of detection models. The toolkit also features a plug-in framework that enables researchers to test, retrain, and compare the performance of different classifiers, fostering continuous improvement in deepfake detection capabilities. Regarding accuracy, specific accuracy metrics are not detailed in the available sources, Deepstar's comprehensive dataset and modular design suggest a robust foundation for developing and evaluating deepfake detection models. The toolkit's structure allows for continuous improvement and adaptation to emerging deepfake techniques. Deepstar has

demonstrated practical utility in real-world scenarios. At the Black Hat security conference, ZeroFox researchers presented their techniques for identifying deepfake videos using Deepstar, highlighting its effectiveness in combating deepfake threats. The toolkit's plug-in framework and curated video library have been instrumental in advancing detection capabilities within the cybersecurity community. (zerofox-oss, 2019)

DeepFake-O-Meter (yuezunli, 2020):

https://github.com/yuezunli/deepfake-o-meter?tab=readme-ov-file

DeepFake-O-Meter, developed by the University at Buffalo Media Forensics Lab, is an open-source platform that integrates multiple state-of-the-art detection algorithms to analyze images, videos, and audio for deepfake content. In deployment, DeepFake-o-meter's backend utilizes a computation server with 8 A5000 GPUs for deepfake detection. Users can upload media through a web-based interface supporting formats such as MP4, JPG, and WAV, and select from various detection algorithms based on metrics like accuracy, runtime, or development year. The platform has demonstrated effectiveness in detecting AI-generated content; for instance, it achieved a 69.7% accuracy rate in identifying AI-generated audio in a Poynter test, surpassing other free tools. Despite these limitations, DeepFake-O-Meter serves as a valuable tool for researchers, educators, and developers aiming to analyze and combat deepfake content. (yuezunli, 2020)

FakeFinder (IQTLabs, 2021):

https://github.com/IQTLabs/FakeFinder

FakeFinder, developed by IQT Labs, is an open-source deepfake detection framework that aggregates predictions from six pre-trained deep learning models, including Boken, Selimsef, and NTechLab, which performed well in the Facebook/Kaggle Deepfake Detection Challenge. It focuses on detecting face-swap deepfakes by analyzing facial inconsistencies, blending boundaries, and temporal anomalies in videos. Its modular design features a Dash-based frontend, API integration capabilities, and an AWS containerized backend. However, FakeFinder's performance drops when detecting non-face-swap manipulations, such as cheapfakes and audio deepfakes, due to limited training

diversity. Additionally, it exhibits racial and gender biases, with higher false positive rates for East Asian faces compared to White faces, attributed to unbalanced training data predominantly featuring White actors. Labeled as a prototype, it has cybersecurity vulnerabilities like unsecured APIs and outdated dependencies, necessitating significant improvements for enterprise use. Despite these limitations, FakeFinder has influenced the development of other detection frameworks and highlighted the importance of addressing biases and ensuring transparency in AI tools. Its open-source codebase continues to serve as a reference for developers exploring multi-model detection systems. (IQTLabs, 2021)

TruFor (grip-unina, 2022):

https://github.com/grip-unina/TruFor

TruFor is a deepfake detection framework that integrates high-level RGB image features with low-level artifacts captured in a learned noise-sensitive fingerprint, effectively detecting and localizing both cheapfakes and deepfakes by identifying anomalies from expected patterns in pristine images. Its transformer-based architecture combines these features, enhancing anomaly detection capabilities, while a reliability map highlights areas where localization predictions may be less reliable, aiding forensic analysis by reducing false positives. Extensive experiments have shown that TruFor outperforms existing methods across various datasets, demonstrating robust generalization to different manipulation techniques. Although archived in 2023, its open-source codebase continues to serve as a reference for developers exploring multi-model detection systems. (grip-unina, 2022)

Comparison:

The following table (Table 3) provides a comparison of several open-source tools and frameworks designed for deepfake detection. These tools vary in terms of supported models, datasets, and key features, offering different strengths for researchers, developers, and cybersecurity professionals. The comparison highlights the core functionalities, performance metrics (where disclosed publicly), and notable aspects of each tool.

Table 3: Overview Comparison of Open-source Tools and FrameWork

Tool/Framework	Supported Models	Datasets/Benchmarks	Accuracy Metrics	Notable Features
DeepSafe	CNNs, GAN-based detectors	FaceForensics++, Celeb-DF	Not disclosed	Supports direct uploads, URL processing, custom model integration, optional GPU acceleration
Deepstar	Not disclosed	Curated library of deepfake and real videos from platforms like YouTube	Not disclosed	Plug-in framework for testing/retraining classifiers, continuous improvement
DeepFake-O-Meter	Multiple detection algorithms	Not disclosed (supports MP4, JPG, WAV formats)	69.7% accuracy (audio detection)	Supports selection of algorithms based on accuracy, runtime, or year, 8 A5000 GPUs for backend
FakeFinder	Boken, Selimsef, NTechLab, etc.	Facebook/Kaggle Deepfake Detection Challenge	Not disclosed	Focus on face-swap deepfakes, Dash-based frontend, API integration, AWS containerized backend
TruFor	Not disclosed	Not disclosed	Not disclosed	Combines high-level RGB features with low-level artifacts, transformer-based architecture, reliability map

2.1.5 Limitations of Existing Deepfake Detection

Generalization Challenges in Cross-Domain Scenarios

A critical limitation of current deepfake detection models is their inability to generalize effectively across diverse datasets and manipulation techniques. For instance, while models like UCF and SPSL achieve high within-domain accuracy (AUC >0.94), their cross-domain performance drops significantly (AUC ~0.77–0.78) (Yan et al., 2023a; Liu et al., 2021). This discrepancy highlights a reliance on dataset-specific artifacts rather than learning universal manipulation patterns. The DF40 dataset, despite incorporating 40 deepfake techniques, still struggles to simulate real-world variability, as models trained on it may fail to detect novel or evolving methods like diffusion-based deepfakes (Yan et al., 2024). Such gaps underscore the need for adaptive architectures that prioritize invariant feature learning.

Bias and Inclusivity Issues in Training Data

Many detection tools exhibit biases rooted in unbalanced training datasets. FakeFinder, for example, demonstrates higher false-positive rates for East Asian faces compared to White faces due to its reliance on datasets dominated by White actors (FakeFinder GitHub documentation). Similarly, spatial detectors like FFD focus heavily on facial regions, rendering them ineffective for non-facial manipulations (Li et al., 2017). These biases not only reduce fairness but also limit practical applicability in global contexts. Tools like Intel's FakeCatcher further face challenges with pixelated or low-quality videos, disproportionately affecting regions with limited bandwidth (Clayton, 2023). Addressing these biases requires more diverse datasets and fairness-aware training protocols.

Computational Overhead and Real-Time Limitations

Several state-of-the-art models, such as UCF and Deepware Scanner, suffer from high computational demands, making them impractical for real-time deployment. UCF's spatial attention mechanisms, while effective, require significant GPU resources, limiting scalability in resource-constrained environments (Yan et al., 2023a). Similarly, DeepFake-O-Meter's reliance on 8

A5000 GPUs for backend processing restricts accessibility for smaller organizations (DeepFake-O-Meter GitHub). While lightweight models like EfficientB4 address this partially, their accuracy trade-offs (AUC ~0.77) highlight a persistent tension between efficiency and performance (Potrimba, 2023).

Overreliance on Specific Artifact Patterns

Frequency-based detectors like SPSL excel at identifying spectral anomalies but risk overfitting to frequency artifacts unique to specific deepfake generation tools (Liu et al., 2021). For example, GAN-generated deepfakes may leave distinct high-frequency noise, whereas diffusion models produce subtler artifacts, evading detection (Zhou et al., 2024). Similarly, tools like Microsoft's Video Authenticator, which focus on blending boundaries, struggle against high-fidelity deepfakes that minimize visual inconsistencies (Kelion, 2020). This narrow focus limits robustness against adversarial attacks or evolving manipulation techniques.

Ethical and Practical Deployment Barriers

Many commercial tools, such as Sensity and Reality Defender, lack transparency in pricing and customization, limiting accessibility for non-commercial users (Sensity, 2023; Reality Defender documentation). Open-source frameworks like Deepstar and FakeFinder, while valuable for research, often suffer from cybersecurity vulnerabilities (e.g., unsecured APIs) and incomplete documentation, hindering enterprise adoption (FakeFinder GitHub). Additionally, tools like iProov's liveness detection, though accurate in controlled settings, face ethical concerns over privacy and potential misuse in surveillance (G-Cloud 13 Service Definition, 2022).

Limited Multimodal Integration

Most tools specialize in single tyle of media analysing, such as video (Deepware Scanner) or audio (Resemble AI's Detect), but fail to address multimodal deepfakes combining audio, video, and text. For example, Sentinel AI processes metadata but does not integrate audio-visual synchronization checks, leaving it vulnerable to lip-sync manipulations (Romain, 2023). Hybrid frameworks like

TruFor, which combine RGB and noise fingerprints, show promise but remain experimental and lack large-scale validation (TruFor GitHub).

2.2 Literature Review of Development Methodology

2.2.1 Introduction

The development of component-based deepfake detection systems necessitates a methodology that accommodates rapid technological advancements, evolving adversarial threats, and complex ethical concerns. Given these challenges, it is essential to select an appropriate development approach that ensures the system remains adaptable, scalable, and ethically sound. This section critically examines four methodologies—Waterfall, Agile, Component-Based Development (CBD), and Spiral—assessing their relevance, strengths, and weaknesses in the context of deepfake detection systems.

2.2.2 Waterfall Methodology

The Waterfall model, introduced by Royce (1970), follows a rigid, linear sequence of phases: requirements gathering, design, implementation, testing, and maintenance. Waterfall is characterized by thorough documentation and structured planning, making it suitable for projects with well-defined, stable requirements. However, its inflexibility renders it less effective for AI projects, particularly deepfake detection systems, where frequent updates and rapid adaptability are crucial.

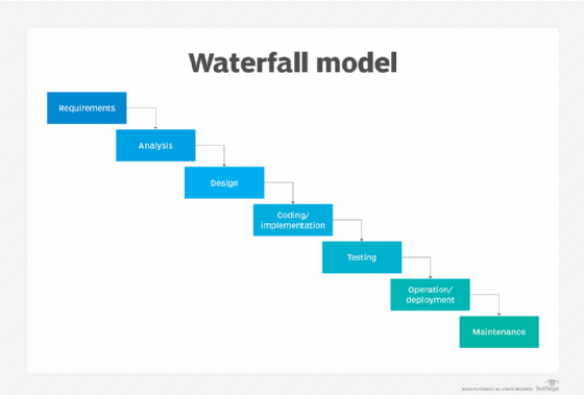

Figure 15: Sample Structure of Waterfall Development Approach (Kirvan, 2022)

Relevance to the Project:

The Waterfall model offers clear milestones and a strong focus on documentation, which can be beneficial during the early stages of dataset curation. For instance, it aligns with the need to catalog images from various deepfake generation models, such as StyleGAN and FaceForensics++, and ensures detailed records of training protocols and model architectures. This structured approach facilitates reproducibility, an important consideration in AI research.

However, The Waterfall model's rigid, sequential structure fundamentally conflicts with the dynamic requirements of a component-based deepfake detection system. In this project, where *modular components* (e.g., StyleGAN validators, auto-encoder analyzers) must adapt incrementally to adversarial advancements like diffusion models or StarGAN architectures, Waterfall's inflexibility becomes a critical bottleneck. Retraining detectors for new architectures would require restarting the entire development cycle—redefining requirements, redesigning connectors, and reimplementing components—rather than leveraging the project's modular framework to update individual units independently. For instance, integrating a diffusion model detector *without disrupting existing components* (e.g., FaceForensics++ validators) is impossible under Waterfall's linear phases, as the methodology lacks mechanisms for iterative refinement.

Furthermore, Waterfall's delayed validation phase, which occurs only after full system implementation, undermines the project's need for *continuous cross-dataset evaluation*. In a component-based system, interoperability between modules (e.g., connector units for output aggregation) must be tested iteratively against diverse datasets (e.g., Celeb-DF, FaceForensics++) to ensure seamless integration. Waterfall's "big bang" testing approach risks late-stage discovery of incompatibilities, such as mismatched input/output formats between GAN artifact detectors and auto-encoder validators, which could derail deployment timelines. This misalignment with modular, incremental development renders

Waterfall unsuitable for maintaining the agility and scalability required to counter evolving deepfake threats.

Discussion: Due to its inflexibility and delayed validation, Waterfall is unsuitable for projects like deepfake detection, where iterative updates and rapid adaptation to emerging techniques are essential.

2.2.3 Agile Methodology

Agile methodologies, including frameworks like Scrum and Kanban, emphasize iterative cycles, stakeholder collaboration, and incremental delivery. These methodologies are particularly suited for projects that require frequent changes and early feedback, such as deepfake detection systems, where new manipulation techniques and datasets are continuously emerging.

Figure 16: Sample Structure of Agile Development Approach (Damm, 2023)

Relevance to the Project:

Agile's rapid prototyping capabilities are critical for the iterative development of modular detection components in this project, such as GAN artifact detectors, frequency analyzers, and auto-encoder validators. By structuring development into 2–4 week sprints, Agile enables the incremental deployment and testing of individual components. For example, a sprint could focus on refining a StyleGAN artifact detector using synthetic data from FaceForensics++, with immediate feedback loops validating its performance on emerging datasets like Celeb-DF v2. This iterative approach ensures that components like the CycleGAN face-swap validator can be refined in parallel, accelerating the system's responsiveness to adversarial advancements like diffusion models.

Agile also fosters cross-functional collaboration, essential for aligning technical and ethical priorities in a component-based framework. During sprint planning, researchers, developers, and ethicists jointly define interfaces for components (e.g., standardized JSON outputs for connector units), ensuring interoperability and ethical compliance from inception. For instance, ethicists might flag biases in the training data for a StarGAN attribute editor detector, prompting immediate dataset adjustments before integration.

While Agile's continuous integration (CI) pipelines automate testing of components against new threats, its reactive risk management struggles with systemic challenges inherent to modular systems. Without standardized interfaces, loosely coupled components risk incompatibility, accumulating technical debt. For example, inconsistent input formats between a CRN-based texture analyzer and a Face2Face expression detector could fragment the system's output aggregation.

Moreover, Agile's sprint-centric focus may overlook proactive mitigation of dataset bias or fairness gaps across components. While a sprint might optimize a proGAN synthesizer detector for accuracy, it may fail to address embedded biases in FaceForensics++ training data, risking skewed performance on underrepresented demographics. This underscores the need to embed Spiral-inspired risk cycles within Agile workflows to ensure ethical rigor alongside rapid iteration.

Discussion: Agile is highly effective for early-stage prototyping and rapid iteration but insufficient for maintaining a scalable, modular system over time, particularly when it comes to managing the systemic risks associated with deepfake detection.

2.2.4 Component-Based Development (CBD)

Component-Based Development (CBD) decomposes systems into independent, reusable modules that can be updated and tested individually. This methodology prioritizes modularity, interoperability, and incremental scalability, making it highly suitable for AI systems like deepfake detection, where adaptability to

new techniques and the ability to integrate diverse components are key requirements.

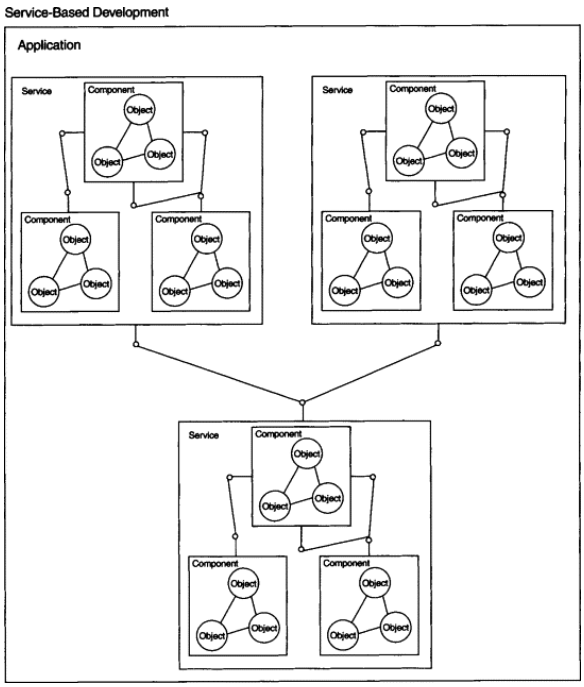

Figure 17: Sample Conceptual Structure of CBD approach (McGovern et al., 2003)

Relevance to the Project:

Component-Based Development (CBD) is central to this project's goal of building a modular, adaptable deepfake detection system. By decomposing the system into specialized, interoperable detection components—such as a StyleGAN artifact detector or an auto-encoder reconstruction analyzer—CBD enables targeted updates without system-wide retraining. For instance, when confronting emerging adversarial techniques like diffusion models, only the diffusion-specific validator requires retraining, while pre-existing components (e.g., CycleGAN face-swap detector) remain unaffected.

However, CBD introduces initial complexity that demands meticulous design. Defining universal input/output formats (e.g., ensuring the auto-encoder validator outputs tensor shapes compatible with the connector unit) requires rigorous cross-component alignment. For example, inconsistencies between the image preprocessing pipelines of the StyleGAN detector (normalized to [-1,

1]) and the auto-encoder validator (normalized to [0, 255]) could corrupt downstream analyses. Additionally, frameworks like TensorFlow lack native support for versioning components, complicating updates. If the StyleGAN artifact detector is upgraded to handle StyleGAN3-generated images, dependency conflicts may arise if the new detector's output tensor shapes (e.g., [B, C, 256, 256]) deviate from the input expectations of downstream components like the frequency-domain analyzer (which expects [B, C, 224, 224]). Without tools like MLflow to track component versions and validate input/output compatibility, this mismatch could corrupt the analyzer's spectral artifact detection.

Discussion: CBD is the most suitable methodology for deepfake detection, as it supports modularity, generalization, and adaptability, enabling targeted updates and integration of new detection components without requiring full system retraining.

2.2.5 Spiral Methodology

The Spiral model, introduced by Boehm (1986), combines iterative development with systematic risk analysis. It progresses through four phases—planning, risk analysis, engineering, and evaluation—repeated in cyclical loops. This methodology is particularly suited for high-risk, complex projects that require continuous risk management, making it a good fit for deepfake detection systems, which must address evolving adversarial techniques and ethical concerns.

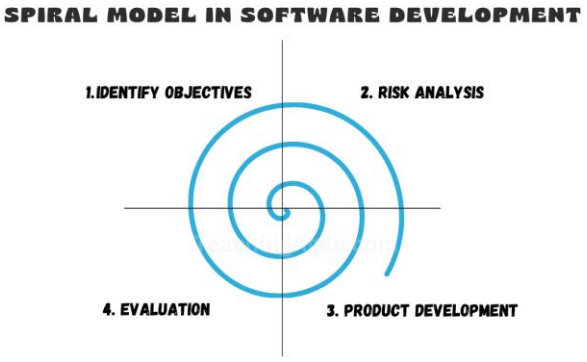

Figure 18: Sample Structure of Spiral Development Approach (Talreja, 2024)

Relevance to the Project:

The Spiral model's risk-driven iterations are uniquely suited to the component-based architecture of this deepfake detection system, where modular components (e.g., StyleGAN validators, auto-encoder analyzers) require continuous refinement against evolving adversarial threats. In Iteration 1, the focus could be on mitigating dataset bias within the StyleGAN artifact detector by curating ethnically diverse training data from FaceForensics++ and Celeb-DF. This ensures the component's robustness across demographics—critical for real-world deployment. Iteration 2 would target generalization failure by refining the CycleGAN face-swap validator and StarGAN attribute editor detector to handle unseen architectures like diffusion models. Cross-component testing would validate interoperability, such as ensuring the frequency analyzer's outputs align with the GAN classifier's input requirements.

Early prototyping in Spiral's engineering phase accelerates the deployment of lightweight, standalone components. For instance, a baseline GAN artifact detector could be prototyped in Iteration 1 using PyTorch, tested on synthetic StyleGAN data, and later expanded in Iteration 2 to include a CRN-based texture validator for detecting perceptual loss-generated forgeries. This phased approach ensures incremental scalability while maintaining system coherence.

Spiral's holistic risk management concurrently addresses technical and ethical risks. During Iteration 3, specific compliance like GDPR could be integrated into the auto-encoder validator's data handling protocols, while fairness audits would evaluate bias in the Face2Face expression detector across gender and ethnicity subgroups. This dual focus ensures ethical rigor without compromising technical performance.

However, Spiral's resource intensity poses challenges for a component-based system. Frequent prototyping of interdependent modules demands robust version control and documentation. For example, retraining the diffusion model validator in Iteration 4 might require backward compatibility checks with the connector unit to avoid system fragmentation.

Discussion: The Spiral model offers significant advantages in risk management and iterative prototyping, making it an effective methodology for addressing the evolving challenges of deepfake detection. However, its resource demands and complexity may pose challenges in terms of project management and timeline.

2.2.6 Comparative Analysis of Methodologies

Table 4 provides a comparative analysis of key development methodologies—Waterfall, Agile, Component-Based Development (CBD), and Spiral—evaluating their suitability for the project's dynamic requirements. This comparison highlights critical trade-offs in adaptability, risk management, and scalability, with CBD and Spiral emerging as strong candidates due to their modular design and iterative risk-assessment capabilities.

Table 4: Comparative Analysis on Different Development Methodologies

Aspect	Waterfall	Agile	Component-Based	Spiral
Generalization Support	***Low*** : Static design fails to adapt to new architectures (e.g., diffusion models).	*Moderate* : Iterative updates improve single components (e.g., GAN detectors).	**High**: Modular components (e.g., StyleGAN validator, auto-encoder analyzer) reuse features across datasets (FaceForensics++, Celeb-DF).	**High**: Risk-driven iterations target cross-architecture gaps (e.g., CycleGAN → StarGAN).
Adaptability	***Poor*** : Requires full retraining for new components (e.g., adding a CRN validator).	**High**: Rapid prototyping of individual modules (e.g., frequency analyzer).	**High**: Independent updates (e.g., upgrading Face2Face detector without affecting GAN validators).	***Moderate*** : Controlled iterations balance new component integration (e.g., diffusion models) with system stability.
Risk Management	***Low*** : Late-stage testing misses component interoperability issues.	***Low*** : Reactive handling of biases (e.g., ethnic imbalance in training data).	***Moderate*** : Modular isolation limits system-wide failures but lacks ethical audits.	**High**: Proactive risk mitigation (e.g., quarterly GDPR audits for connector units).
Real-World Deployment	***Delayed*** : "Big bang" deployment risks incompatibility (e.g., mismatched APIs).	***Fast but fragmented*** **:** Components (e.g., audio deepfake detector) lack cohesion.	**Structured**: Standardized connectors (e.g., REST/JSON APIs) ensure seamless integration.	**Controlled iterations**: Phased rollout (e.g., validate StyleGAN detector → auto-encoder → diffusion model).
Resource Demand	***Low*** : Minimal overhead but incompatible with dynamic deepfake evolution.	***Moderate*** : Sprint cycles require team coordination but reduce rework.	***Moderate*** : Upfront design for component interfaces (e.g., TensorFlow SavedModel formats).	**High**: Frequent prototyping (e.g., testing CRN validators) and risk analysis (e.g., fairness metrics).

2.2.7 Summary:

The component-based deepfake detection project requires a methodology that can effectively balance rapid adaptability, robust risk management, and modular scalability. After evaluating various development approaches, including traditional Waterfall, Agile, Component-Based Development (CBD), and Spiral methodologies, the decision was made to adopt a hybrid Agile-Spiral approach. This decision is motivated by the need to leverage Agile's rapid iteration and prototyping capabilities, while also integrating Spiral's systematic risk management to ensure that both technical and ethical challenges are addressed in a structured manner. Additionally, incorporating CBD principles ensures the system's modularity, scalability, and generalization across diverse datasets and adversarial techniques.

Table 5: Rating of Hybrid Agile-Spiral Approach in different aspect

Aspect	Hybrid Agile-Spiral
Generalization Support	**Very High** (modular + risk-driven)
Adaptability	**Very High** (Agile + CBD modularity)
Risk Management	**Very High** (Spiral cycles + ethics)
Real-World Deployment	**Structured + Controlled** (CI/CD + risk phases)
Resource Demand	**High** (balanced via Agile efficiency)

The hybrid Agile-Spiral framework combines key features from both methodologies. Agile sprints, which typically last 2–4 weeks, will focus on rapidly developing and prototyping individual detection components, such as GAN artifact detectors and frequency analyzers. Continuous integration and deployment (CI/CD) tools will be used to automate the testing of these components against new datasets and models, ensuring that the system stays up-to-date with evolving deepfake techniques. At the same time, Spiral cycles will guide the project through quarterly risk assessments, addressing both technical

risks and ethical concerns. The combination of rapid prototyping and risk management ensures a comprehensive, adaptable development process.
The use of Component-Based Development (CBD) further supports the hybrid approach by ensuring the deepfake detection system remains modular and scalable. By developing decoupled components, such as an auto-encoder validator or an audio deepfake detector, the system can be updated incrementally as new models and techniques are introduced. Connector units, such as standardized APIs, ensure that these components work seamlessly together, allowing for easy integration of new modules without disrupting the overall system.

Such hybrid methodology offers several benefits, including enhanced adaptability, improved generalization across datasets, better ethical compliance, and scalability for cloud-based deployment. The implementation roadmap is divided into three phases, each aligning Agile sprints and Spiral cycles. In the first phase, the focus will be on developing a baseline detector and mitigating dataset bias. In subsequent phases, cross-architecture performance will be evaluated, and ethical audits will ensure the system meets compliance standards. In conclusion, the hybrid Agile-Spiral approach is the optimal strategy for this deepfake detection project, combining the flexibility of Agile, the structured risk management of Spiral, and the modularity of CBD. This approach will ensure the system remains adaptable, scalable, and ethically sound as it evolves to meet the challenges of deepfake detection.

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 METHODOLOGY

3.1.1 Introduction

This methodology outlines the systematic process for developing a ensemble deepfake detection system, emphasizing user-centric design, modular architecture, and ethical compliance. The workflow is divided into three core phases: Front-End Development, Back-End Development, and System Integration. Each phase ensures that the system remains adaptable, scalable, and ethically sound, addressing the needs of end-users while incorporating continuous feedback loops.

3.1.2 Stage 0: Exploration, Prototyping, and Model Training

The process commenced with Stage 0, where baseline models were deployed and trained to establish comparative performance benchmarks. Initial feasibility tests were carried out in a Jupyter Notebook environment (model_deployment.ipynb on Google Colab), where multiple architectures — including Xception, CapsuleNet, and YOLO — were instantiated to validate the viability of deepfake detection across diverse model families. In parallel, dedicated training was performed for four selected detectors: EfficientNet-B4, Meso4, MesoInception, and UCF. These model are selected due to high accuracy and performance in review of Deepfakebench. (May refer back to Literature Review Chapter) Among these, EfficientNet-B4 was integrated as the principal detector within the ensemble framework, while Meso4, MesoInception, and UCF served primarily as baselines for comparative analysis of the final system. Additional pretrained detectors were incorporated from team contributions, thereby broadening the comparative scope. A simple ensemble based on majority voting was also implemented to examine the benefits of model fusion. Deliverables from this stage included reproducible training logs,

hyperparameter records, and baseline evaluation metrics (accuracy, precision, recall) across curated test sets.

3.1.3 Stage 1: Foundational Component Development

With these baselines established, Stage 1, introduced the modular infrastructure required for a scalable system. A unified detector framework (unified_detector.py) was engineered to abstract preprocessing, model loading, inference, and result serialization into a consistent API contract. This was later extended into a configuration-driven variant that consumed YAML/JSON specifications, enabling detectors to be registered and instantiated dynamically without code-level changes. Complementing this, single-model wrappers (single_model_detector_wrapper.py) were introduced to encapsulate individual detectors, ensuring isolated execution and streamlined benchmarking. This stage also introduced hyperparameter tracking systems, which ensured that model configurations (e.g., optimizer choice, learning rate schedules, batch size) were systematically documented and could be replicated across experiments. Exit criteria for this stage included successful encapsulation of representative models and validation of standardized output schemas.

3.1.4 Stage 2: Hardened Deployment and Orchestration

Stage 2 concentrated on system reliability and engineering discipline. No new model training was introduced at this stage; instead, emphasis was placed on transforming the system into a robust service layer. Key engineering practices included the integration of Continuous Integration/Continuous Deployment (CI/CD) pipelines on Git Action to automate testing and ensure reproducibility, as well as the implementation of validation harnesses that enforced correctness across detectors. Fault-tolerant orchestration mechanisms were introduced to allow concurrent inference requests, supported by asynchronous execution strategies that minimized latency under load. Performance benchmarks were systematically defined, including sub-500ms average inference latency per image, throughput exceeding 1,000 requests per day, and resilience to malformed or corrupted inputs. Spiral reviews at this stage prioritized risk

reduction in three domains: scalability under concurrent requests, resilience to adversarial perturbations, and maintainability of code across multiple contributors.

3.1.5 Stage 3: Ensemble Aggregation and Cross-Domain Evaluation

In Stage 3: Ensemble Aggregation and Cross-Domain Evaluation, the platform advanced from a collection of isolated detectors into an integrated ensemble system. Ensemble v1 provided a minimal baseline with sequential and parallel voting, while Ensemble v2 introduced a configuration-managed aggregation framework capable of performing confidence-weighted fusion, adaptive thresholding, and uncertainty estimation. This evolution culminated in the ensemble_detector package, a production-grade framework encapsulating dynamic routing, advanced logging, structured error handling, and performance monitoring. The most significant addition at this stage was the development of an API server module, which exposed the ensemble detection capabilities through RESTful endpoints. These endpoints adhered to a contract-driven design, accepting JSON-formatted image payloads and returning structured responses including prediction labels, confidence scores, optional heatmaps, and standardized error codes. By establishing this API, the ensemble detector was transformed from a research prototype into a consumable service aligned with service-oriented architecture principles.

3.1.6 Stage 4: Modularization, Productionization, and Integration

Finally, Stage 4: Modularization, Productionization, and Integration completed the transformation of the platform into a deployable system with front-end accessibility. The detector_output_wrapper module enforced output standardization across JSON, XML, and Python dictionary formats, while the ensemble_detector package acted as the inference engine exposed via its API server. Integration was realized through a Laravel web application, which operated as the user-facing interface. The Laravel system consumed the API endpoints, authenticated users, and enforced role-based access control. General users interacted with the platform through a simplified drag-and-drop interface

that returned binary “real/fake” classifications, with visualized component-level metrics. Administrators could activate or deactivate detection modules directly through the Laravel interface, effectively managing system configurations at runtime without requiring redeployment. Production readiness was reinforced through observability features, including log aggregation, error alerting, performance dashboards, and rollback procedures. In addition, ethical safeguards were embedded into the production pipeline, including demographic fairness auditing, which ensured that disparities in false positive rates across different demographic groups did not exceed five percent. Spiral reviews continued at this stage to reassess new risks, such as the emergence of novel deepfake generation methods and the accumulation of technical debt.

3.2 Project Work Plan

3.2.1 Introduction

This project work plan is designed to deliver a robust, ethical, and scalable deepfake detection system utilizing ensemble methods with a strong focus on modularity, flexibility, and performance. The project follows a hybrid Agile-Spiral methodology, ensuring controlled increments, continuous feedback, and risk management. The work plan is organized into four distinct phases, each with a clear focus, milestones, and risk management strategies.

3.2.2 Phase 1: Front-End Development (Weeks 1–4)

Objective: Build a user-centric interface that is intuitive, accessible, and compliant with ethical standards. The front-end will be developed in an iterative process with continuous user feedback and usability testing.

Key Features:

- **User Interface (UI) Design**: The UI will provide an easy-to-use platform for both technical and non-technical users, ensuring accessibility.
- **Iterative Development**: Regular usability testing and feedback will be incorporated to improve user experience.

- **Ethical Transparency**: Clear information on data usage, system functions, and model explanations will be provided to the users.

Milestone 1:

- Prototype of the UI ready for internal review and feedback.
- **Risk Assessment**: Week 4 review to assess UI usability, ethical transparency, and potential biases in design.

Table 6: Key Element in (Phase 1)

Task	Details	Timeline
UI Design and Implementation	Initial prototype based on user feedback and iterative testing	Week 1–3
Ethical Transparency Integration	Ensure all relevant ethical data and model behavior is clearly communicated to users	Week 2–3
Feedback and Usability Testing	Continuous user feedback, adjustments, and refinement	Week 3–4

3.2.3 Phase 2: Back-End Development (Duration: Weeks 5–9)

Objective: Develop modular detection components for the system. This phase focuses on building the core back-end infrastructure and ensuring model validation and version control.

Key Features:

- **Modular System**: Independent model wrappers for different deepfake detection models, with standardized input and output formats.
- **Version Control and Risk Management**: Rigorous validation and testing procedures to ensure robustness and performance.
- **Ensemble Integration**: Assemble multiple detection models into an ensemble system for improved accuracy and resilience.

Milestone 2:

- Completion of back-end components, with a fully operational pipeline ready for integration.
- **Risk Assessment**: Week 9 review to evaluate model robustness, fairness, and potential technical debt.

Table 7: Key Element in (Phase 2)

Task	Details	Timeline
Model Wrapper Development	Build and integrate wrappers for individual models (e.g., YOLO, Xception, EfficientNet)	Week 5–7
Model Validation and Testing	Evaluate models on performance benchmarks (accuracy, precision, recall, etc.)	Week 6–8
Ensemble System Development	Implement initial ensemble model	Week 7–9

3.2.4 Phase 3: System Integration (Duration: Weeks 10–12)

Objective: Integrate all the system components into a unified platform. The focus will be on finalizing the back-end and front-end integration, ensuring performance benchmarks, and final risk assessments.

Key Features:

- **Seamless Integration**: Integration of back-end detection models with the front-end Laravel UI for real-time user interaction.
- **Risk Monitoring**: Continuous monitoring of system performance, identifying any potential issues such as technical debt and system vulnerabilities.
- **Final Testing and Deployment**: Ensure the system meets all functional and non-functional requirements, including scalability and robustness.

Milestone 3:

- Deployment of the complete system for end-user access.
- **Risk Assessment**: Final audit of ethical compliance, system stability, and technical debt management.

Table 8: Key Element in (Phase 3)

Task	Details	Timeline
Back-End and Front-End Integration	Integrating ensemble model, backend logic, and front-end UI	Week 10–11
Performance Optimization	Ensure the system meets performance benchmarks (throughput, inference time, scalability)	Week 11
Final Testing and Deployment	Test all system components, fix bugs, and deploy the final version for user access	Week 12

3.2.5 Expected Project Tools:

The selected development tools directly support the iterative refinement of deepfake detection components and the risk-driven validation cycles outlined in the work plan. These tools are integral to the system's development, ensuring alignment with the objectives of accuracy, scalability, and ethical compliance. Below is an overview of the tools and their applications in the project

Table 9: Table of Expected Tools Involved in Development

Tool	Purpose	Application in the Project
TensorFlow / PyTorch	Deep learning frameworks for building and training detection models	Facilitate rapid prototyping of deepfake detection models. These frameworks power the individual models within the ensemble system, ensuring robust and efficient model training.
Google Colab / Kaggle	Cloud-based notebooks for collaborative development and experimentation	Provide an interactive development environment where models are trained and tested in real-time. Supports easy experimentation with different deepfake detection architectures.
Visual Studio Code	Code editor for writing, testing, and debugging the code	Used for writing and editing Python scripts, model implementation, and managing version control via Git. Provides a rich environment for development with support for various plugins.
Laravel	PHP framework for backend development and API integration	Used for building the backend of the project, handling server-side logic, model management, and user authentication. Integrates with the deepfake detection system through APIs.

Docker	Containerization platform for packaging applications and environments	Ensures consistent environments across development, testing, and production. It packages the models and dependencies into containers for easy deployment and scalability.
GitHub	Version control and collaboration platform	Manages the source code and enables collaboration among team members. Tracks code changes and provides integration with Continuous Integration (CI) systems.
TensorBoard	Visualization toolkit for tracking model metrics and performance during training	Helps visualize training metrics such as loss and accuracy during model development. Assists in monitoring model improvements and comparing performance across various experiments.
Laravel (Frontend Integration)	Framework for frontend API interaction	Laravel provides backend management for the user interface where the deepfake detection results are displayed, ensuring seamless integration with frontend components.

3.3 System Design and Requirements

3.3.1 Introduction

The preliminary result of this project outline the system specifications, ensuring alignment with the core objectives of generalization, accessibility, and scalability in deepfake detection including functional and non-functional requirements, use case descriptions, and architectural workflows that define the proposed web platform's capabilities.

3.3.2 Project Specification

3.3.2.1 Functional Requirement:

Table 10: Functional Requirements

FR Code	Requirement	Description	Priority
FR-001	**User Authentication and Role Management**	The system distinguishes between Normal Users and Administrators. Normal users can access detection functions, while administrators manage models and monitor performance.	High
FR-002	**Image Upload and Validation**	Users upload images (JPEG, PNG, ≤4 MB). Uploaded files are validated for type, size, and resolution. Invalid inputs return meaningful error messages.	High
FR-003	**Single-Model and Ensemble Detection**	Users can select either a single detection model or an ensemble of models. The chosen model(s)	High

		process the image, and results are returned with confidence scores.	
FR-004	**Detector Execution**	Each model wrapper preprocesses inputs, performs inference, and produces standardized outputs (label, confidence, inference time).	High
FR-005	**Ensemble Aggregation**	The ensemble engine supports majority voting and confidence-weighted strategies, combining predictions from multiple detectors. The system tolerates failure of individual models.	High
FR-006	**Result Presentation**	The Laravel front end displays detection results (“real” or “fake”) with confidence scores. If ensemble mode is selected, both per-model outputs and the aggregated ensemble result are shown.	High

3.3.2.2 Non-functional Requirement:

Table 11: Non-Functional Requirements

NFR Code	Requirement	Description	Priority
NFR-001	**Performance**	Images must be processed within ≤800 ms on average and ≤1200 ms at the 95th percentile.	High
NFR-002	**Accuracy and Generalization**	Ensemble detection must achieve ≥90% F1-score on benchmark datasets and not lose	High

		more than 5% AUROC in cross-domain evaluations.	
NFR-003	**Scalability**	The system must handle ≥1,000 daily requests with 99.9% uptime, supporting deployment in containerized environments.	High
NFR-004	**Reliability**	The system must return results even if one or more models fail, using retries and timeouts for resilience.	High
NFR-005	**Security**	File uploads must be sanitized, HTTPS must secure communications, and user data must not persist beyond inference.	High
NFR-007	**Maintainability**	New models can be integrated via configuration files without altering core code. APIs must comply with OpenAPI 3.0.	Medium
NFR-008	**Usability**	The interface must remain simple and intuitive, allowing non-technical users to select detection type (single model vs ensemble) and view clear outputs.	Medium

3.3.2.3 Use case Diagram

This use case diagram (Figure 25) visualizes the core interactions between users and the proposed deepfake detection system, encapsulating key functionalities such as media upload, component selection, and report generation. By mapping roles (e.g., General User, Admin) to system capabilities, this diagram clarifies how the platform balances accessibility for non-experts with advanced controls for administrators.

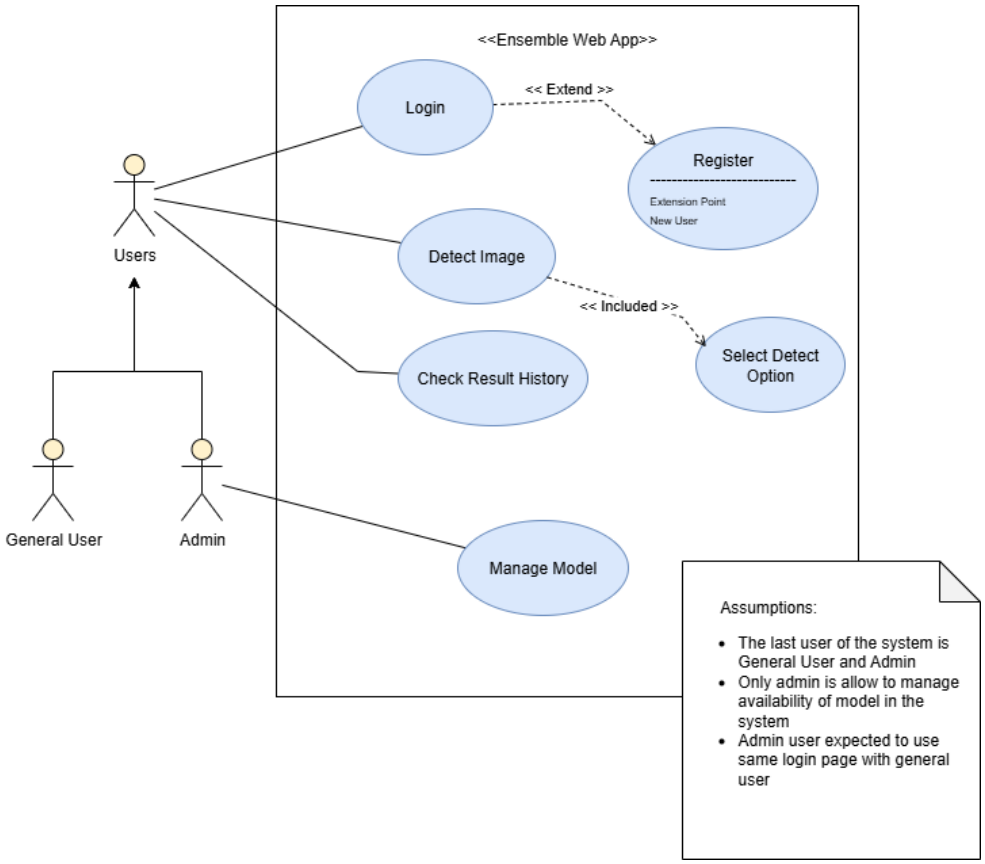

Figure 19: Use Case Diagram

3.3.2.4 Use Case Description:

The use case descriptions in this section expand on the interactions outlined in the use case diagram (Figure 25), providing granular insights into system workflows such as user authentication, media processing, and component management. By detailing step-by-step scenarios—from General Users uploading images for detection to Admins integrating new modules.

Use Case Name**: Login**	ID**: UC-001**	Importance Level**: high**
Primary Actor**: User (General User/Admin)**	Use Case Type**: Brief, Real**	

Brief Description**:** Users (both general and admin) must log in to the system to access the functionalities based on their role. The system checks the credentials, grants access and logs them in to the appropriate interface.
Precondition: The user must have registered an account. The user is on the login page.
Postcondition: The user is authenticated and logged in, redirected to their respective home page or dashboard based on their role.
Relationships: Association **: User** Include **: n/a** Extend **: Register Account (UC-002)** Generalization**: n/a**
Normal Flow of Events**:** 1. **User Navigates to Login Page**: The user opens the login page of the web application. 2. **User Enters Credentials**: The user enters their username and password in the login form. 3. **System Verifies Credentials**: 3.1 The system checks the entered username and password against the stored data. 3.2 If the credentials are correct, the system proceeds to step 4. 3.3 If the credentials are incorrect, the system displays an error message (see Exception Flow). 4. **System Authenticates User**: The system authenticates the user, assigns the correct role (General User or Admin), and grants access to the system. 5. **System Redirects User**: The user is redirected to their respective home page based on their role (either General User Dashboard or

Admin Dashboard).
Sub-flows**:** -
Alternate/Exceptional Flows: • **Invalid Credentials**: If the user enters incorrect login details: o The system will display an error message like "Invalid username or password". o The user can try again with correct credentials. • **Network Issue**: If the system cannot connect to the database due to a network error, an error message "Network error, please try again" will be displayed, and the user must retry later.

Use Case Name**: Register**	ID**:** UC-002	Importance Level**: high**
Primary Actor**: User (General User)**	Use Case Type**: Brief, Real**	
Brief Description**:** A new user registers by providing necessary details. After registration, they can log in and access the system.		
Precondition: The user is not already registered. The user is on the registration page.		
Postcondition: The user account is created, and they are redirected to the login page to enter their credentials.		
Relationships: Association **: User** Include **: n/a**		

Extend **: Register Account (UC-002)**

Generalization**: n/a**

Normal Flow of Events**:**

1. **User Navigates to Register Page**: The user clicks on the "Register" link and is taken to the registration page.
2. **User Enters Registration Details**: The user fills in the required fields like name, email, password, and other necessary details.
3. **System Validates Input**: The system checks if all required fields are filled and if the email is valid.

   3.1 If the inputs are valid, the system moves to step 4.

   3.2 If any input is invalid (e.g., missing field or invalid email format), the system prompts the user to correct the error.
4. **System Creates User Account**: The system creates a new account in the database with the entered details and stores the user's credentials securely.
5. **System Redirects to Login Page**: The user is redirected to the login page with a message confirming that their account has been created.

Sub-flows**:**

-

Alternate/Exceptional Flows:

Alternative Flow(s):

- Email Already Registered: If the user attempts to register with an already used email:
  - The system will show a message like "This email is already registered" and prompt the user to use a different email or log in if they already have an account.

Exception Flow(s):

- System Error During Registration: If there is an issue with the database or server while creating the account, an error message will be displayed (e.g., "An error occurred while creating your account, please try again later").

Use Case Name**:** **Detect Image**	ID**:** UC-003	Importance Level**: high**
Primary Actor**:** User [General User, Admin]	Use Case Type**: Detail, Real**	
Brief Description**:** The user uploads an image to be analyzed by the deepfake detection model. The system processes the image and returns a prediction with a confidence score.		
Precondition: The user must be logged in and on the detection page.		
Postcondition: The user must be logged in and on the detection page.		
Relationships: Association **: User** Include **: n/a** Extend **: n/a** Generalization**: n/a**		
Normal Flow of Events**:** 1. **User Selects "Detect Image" Option**: The user selects the option to detect deepfakes from the available dashboard or menu. 2. **User Uploads Image**: The user selects and uploads an image file (JPEG, PNG). 3. **System Validates Image**: The system checks the file type and size. If the image is invalid (e.g., too large, wrong format), the system prompts the user to upload a valid image.		

4. **System Preprocesses Image**: The image is passed through a preprocessing pipeline (e.g., resizing, normalization) to make it ready for model input.
5. **User Select Detect Choice**: Single model or ensemble, if single model selected, user will be prompted to choose which model to use
6. **System Detects Deepfake**: The system runs the preprocessed image through the selected model(s) and returns the prediction result, including the label ("REAL" or "FAKE") and confidence score.
7. **System Displays Result**: The result is displayed to the user on the front end, showing the prediction label, confidence score, and any additional relevant information (e.g., model used).

Sub-flows**:**

-

Alternate/Exceptional Flows:

Alternative Flow(s):

- **Invalid Image Format or Size**: If the uploaded image doesn't meet the size or format requirements:
    - The system prompts the user to upload a valid image with the appropriate format and size.

Exception Flow(s):

- **Model Processing Error**: If the model fails during inference (e.g., due to a corrupted model or system crash), an error message is displayed, and the user is instructed to try again later.

<table>
<tr><td colspan="2">Use Case Name: Check Result History</td><td>ID: UC-00</td><td>Importance Level: high</td></tr>
<tr><td colspan="2">Primary Actor: User [General User, Admin]</td><td colspan="2">Use Case Type: Detail, Real</td></tr>
<tr><td colspan="4">Brief Description: Users can view their previously uploaded images along with the results of deepfake detection.</td></tr>
<tr><td colspan="4">Precondition: The user must be logged in, and there should be previously processed images in the system.</td></tr>
<tr><td colspan="4">Postcondition: The user can view a list of past detection results, including the images and corresponding predictions.</td></tr>
<tr><td colspan="4">Relationships:
Association : User
Include : n/a
Extend : n/a
Generalization: n/a</td></tr>
<tr><td colspan="4">Normal Flow of Events:
1. User Selects "Check Result History" Option: The user navigates to the history section on the dashboard.
2. System Retrieves Historical Data: The system queries the database for the user's previous detection results.
3. System Displays Results: The system displays the past results, including the image and its predicted label (REAL/FAKE) along with the confidence score.</td></tr>
<tr><td colspan="4">Sub-flows:
-</td></tr>
<tr><td colspan="4">Alternate/Exceptional Flows:
Alternative Flow(s):</td></tr>
</table>

- **No Previous Results**: If the user has no previous detection results, the system will display a message like "No results found" or prompt the user to upload an image for detection.

Exception Flow(s):

- **System Error During Data Retrieval**: If the system encounters a problem while fetching results (e.g., database issues), an error message will be shown.

Use Case Name**:** **Manage Model**	ID**:** UC-005	Importance Level**:** **high**
Primary Actor**:** Admin	Use Case Type**: Detail, Real**	
Brief Description**:** The admin manages the deepfake detection models in the system, including enabling, disabling, or updating model configurations		
Precondition: The user must be an Admin, and the system should have at least one model in the system.		
Postcondition: The system's model configurations are updated accordingly.		
Relationships: Association **: Admin** Include **: n/a** Extend **: n/a** Generalization**: n/a**		
Normal Flow of Events**:** 1. **Admin Selects "Manage Model" Option**: The admin navigates to the model management section. 2. **Admin Modifies Model Configurations**: The admin can enable, disable, or update, upload new model file, weight or delete the configurations of available models.		

3. **System Updates Model Configurations**: The system saves the changes to the model configurations, which could include updating the model's path, enabling/disabling it, or changing its processing parameters. 4. **System Confirms Changes**: The system confirms the success of the update and applies changes to the active model configurations.
Sub-flows**:** -
Alternate/Exceptional Flows: **Alternative Flow(s):** • **Invalid Configuration Input**: If the admin provides incorrect configuration details, an error message will be shown, and the system will prompt the admin to correct it. **Exception Flow(s):** • **Access Denied**: If a non-admin user attempts to access the model management page, an "Access Denied" message will be shown.

3.3.3 High Level System Flow Diagram:

The high-level system flow diagram (Figure 26) synthesizes the use case scenarios and technical specifications into a cohesive visual blueprint, illustrating the end-to-end workflow of the deepfake detection platform. From user-initiated media uploads to backend processing via modular components, this diagram clarifies how data traverses the system, emphasizing critical decision points such as dataset validation.

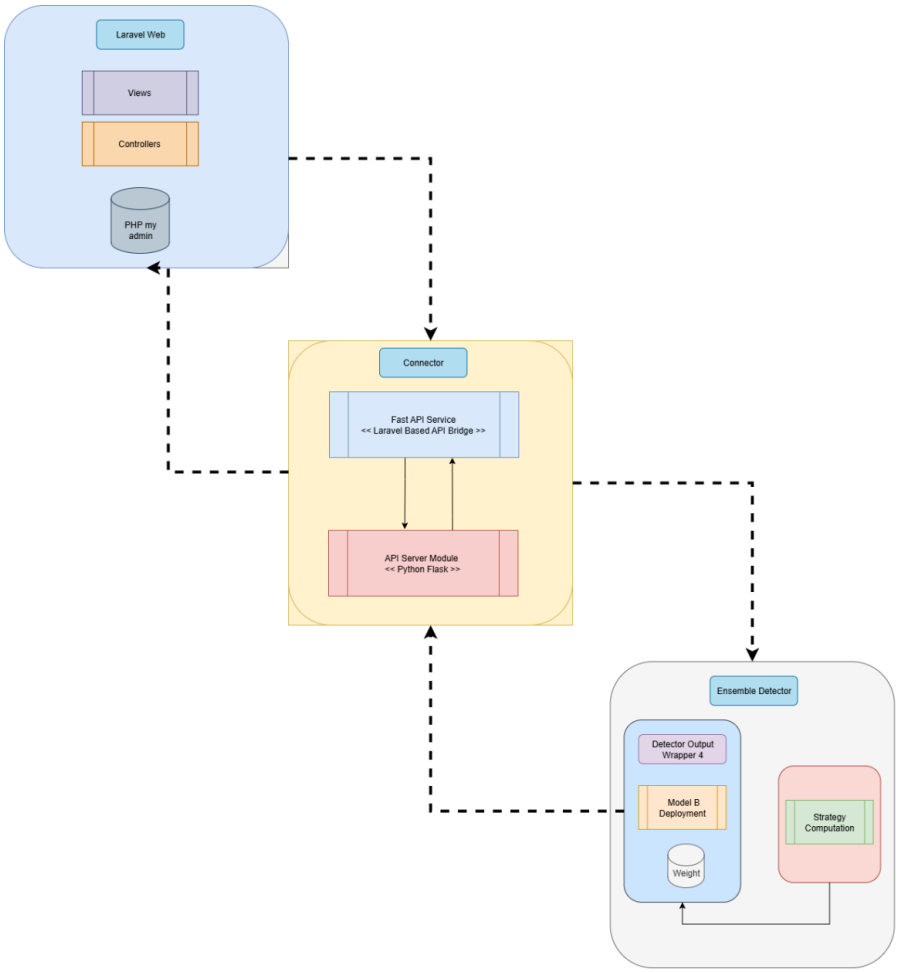

Figure 20: High Level System Flow Diagram

3.3.4 System Architecture Diagram

The System Architecture Diagram provides a detailed view of the structure and interaction flow of the Deepfake Detection System. It is organized into several distinct layers, each serving a specific function in the overall architecture.

Key Components:

Frontend Layer:

The React Component handles the user interface (UI), enabling users to interact with the application. It communicates with the backend API to request image uploads, predictions, and detection results. The View

represents the visual output, displaying results and providing interaction points for users to upload files and adjust settings.

Backend Layer:
The Laravel Router manages API routing by directing requests to the appropriate controllers such as DetectionController, AdminController, and UserController, while the Authentication Middleware ensures secure access by verifying user identity and roles (user/admin).

Business Logic Layer:
The Detection Service powers the core detection process by working with both image processing and machine learning components, while the Model Management Service oversees deepfake detection models, handling tasks like loading configurations, retraining, and updates. The User Service supports user-related functions, including managing data, authentication, and role assignments.

ML Processing Layer:
The Ensemble Detector System serves as the main machine learning engine for deepfake detection, leveraging multiple models like Xception, EfficientNet, and YOLO. Supporting it, the Preprocessing Pipeline prepares images through steps such as resizing and normalization, while the DataLoader ensures proper loading and preparation of image data. A Wrapper provides a unified interface to integrate various detection models seamlessly into the ensemble, and the Ensemble API Server manages prediction logic, combining results from different models to deliver final outputs.

Data Layer:
The MySQL Database manages structured data such as user information, uploaded image metadata, model configurations, detection results, and

system events, while the File Storage system is responsible for storing uploaded image files and model weights.

This architecture ensures a modular and scalable deepfake detection system that integrates various components, such as machine learning models, user management, and detection workflows, while maintaining high performance and flexibility for future updates and improvements.

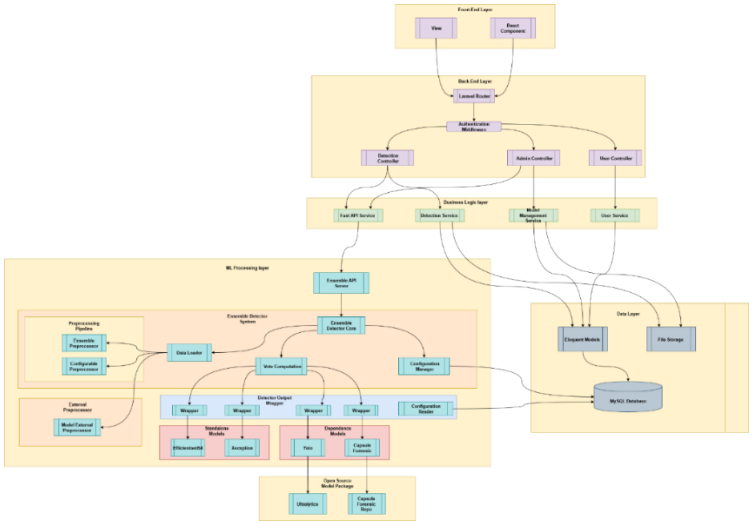

Figure 21: System Architecture Diagram

3.3.5 ERD diagram (Laravel Web Application)

This Entity Relationship Diagram (ERD) illustrates the key relationships and structure of the database for the Deepfake Detection System. The ERD provides an overview of how different entities in the system, such as Users, Uploaded Images, Detection Results, Model Configurations, and System Events, interact with each other.

Key Entities:

Users: Stores information about the system's users, including their roles (e.g., admin, general user).

Uploaded Images: Contains metadata for each image uploaded by users for detection, including file information and storage paths.

Detection Results: Holds the output of the deepfake detection process, including prediction results, model details, and confidence scores.

Model Configs: Contains configuration data for individual machine learning models, such as model type, parameters, and settings for each detection model.

System Events: Tracks system-related events like model performance, system errors, or actions triggered by users for auditing and monitoring purposes.

Relationships:

Users in the system can upload multiple images and generate multiple detection results, establishing a direct link between each user, their uploaded images, and the corresponding outcomes. Uploaded images are analyzed and tied to detection results, ensuring that every image has a clear detection outcome. Each detection result is further connected to single or a list of model configuration, indicating which models were used for analysis. Additionally, system events are tracked and associated with users to monitor actions and events triggered within the system, providing a complete view of user activity and system interactions.

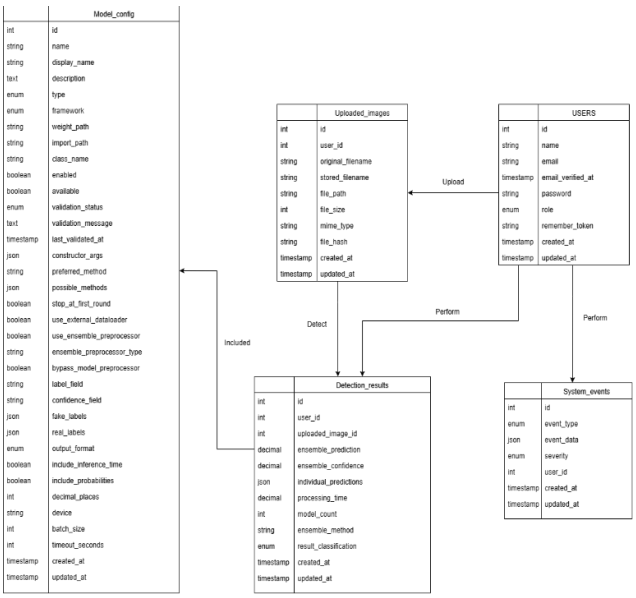

Figure 22: ERD Diagram

3.3.6 Image Detection Sequence Diagram

This sequence diagram illustrates the end-to-end workflow of the deepfake detection system. It begins with the user uploading an image through the web interface, which triggers the Laravel API to handle the request. The image file is stored in file storage, while metadata is saved in the database. Once stored, a detection request is initiated, passing the image to the Ensemble API. The Ensemble System coordinates multiple models to perform predictions, aggregates the results, and returns a consolidated detection outcome. Finally, the Laravel API saves the detection result in the database and delivers the response back to the user interface for result display.

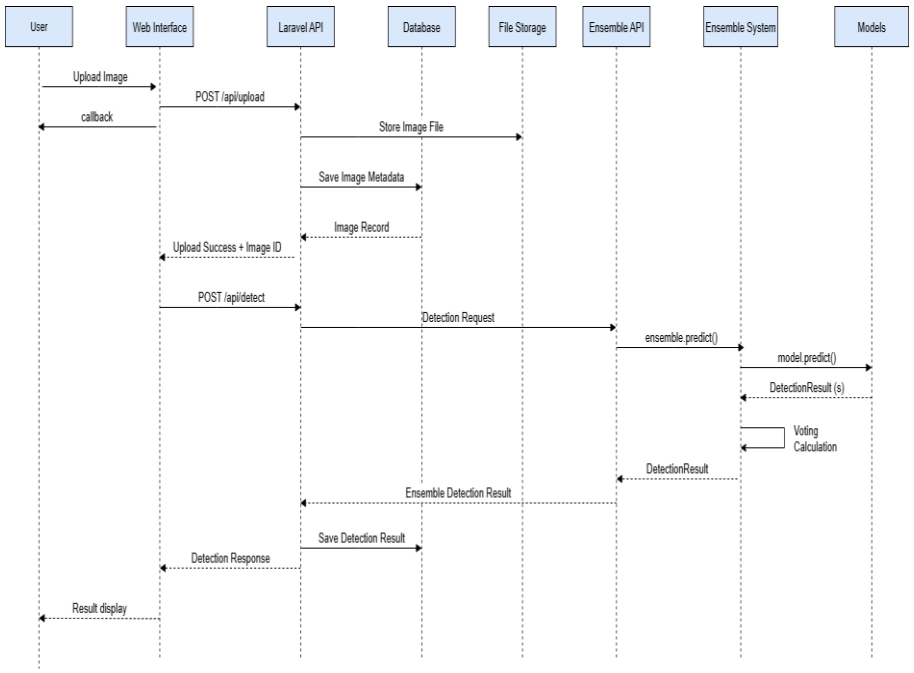

Figure 23: Detect Image Sequence Diagram

3.4 Test Plan

3.4.1 Introduction

The testing strategy for the ensemble deepfake detection system was developed with the explicit aim of ensuring that every implemented feature was rigorously validated. Rather than testing hypothetical or future features, the focus remained on the actual codebase, thereby aligning the evaluation with the project's scope and objectives. The test plan was designed as a multi-layered process, moving from fine-grained unit tests through integration testing to a final end-to-end validation stage. In doing so, the strategy provides assurance that individual modules operate correctly in isolation, that these modules interact smoothly when combined, and that the complete system behaves reliably under realistic usage scenarios. Importantly, the test plan also covers both functional and non-functional requirements, with attention to accuracy, latency, robustness, and fairness.

3.4.2 Objectives

The test plan aims to confirm that all functional modules of the system operate correctly under both normal and adverse conditions. It seeks to verify that system components integrate seamlessly, ensuring reliable end-to-end workflows. Additionally, the plan focuses on ensuring that performance goals, such as latency, throughput, and scalability, are met. The robustness of the system is also tested, particularly in handling errors, corrupted inputs, and concurrent requests. Finally, the test plan aims to demonstrate that the system's outputs meet the expected standards of accuracy, fairness, and usability, ensuring the platform performs as intended across a variety of scenarios.

3.4.3 Test Suite Summary

Total Tests Implemented: 44

Table 12: Test Suite Summary

Category	Count	Coverage Focus

Unit Tests	35	Core modules (wrappers, configs, output formatting, ensemble aggregation)
Integration Tests	3	Cross-module orchestration, configuration-driven workflows, model integration
Stress Tests	1	Full system validation under production-like stress, including load and scalability
Adversarial Tests	5	System robustness against compression, noise, format manipulation, and ensemble disruptions

3.4.3.1 Unit Testing

Unit testing formed the backbone of the test framework, accounting for thirty-six test cases across seven critical modules. These tests were essential for validating correctness at the function and class level, ensuring that each building block of the system performed as intended. The Detector Output Wrapper, for example, required particularly thorough validation because of its role in harmonizing outputs from diverse models. Similarly, modules such as Configuration Management and DetectionResult were scrutinized to guarantee resilience against invalid inputs, schema mismatches, and serialization errors.

The table below summarizes the unit test suites and their coverage:

Table 13: Summary of Unit Test Cases

Suite	Tests	Coverage Focus
Detector Output Wrapper	TC-DOW-001 to TC-DOW-008	Initialization, configuration loading, method detection, output conversion, error handling, metadata extraction,

		prediction invocation correctness, configuration integration (ensures proper behavior across model types and configuration formats).
Detection Result	TC-DR-001 to TC-DR-008	Data structure validation, JSON and dictionary serialization, confidence score validation, timestamp handling, metadata preservation, integrity checks, and output detail levels (ensures correct formatting for JSON, CSV, and other export formats).
Configuration Management	TC-CM-001 to TC-CM-008	YAML loading, schema compliance validation, model instantiation from configuration, configuration persistence, error handling for invalid configurations, environment variable overrides, fallback handling for incomplete configurations, and nested configuration structures.
Ensemble Strategies	TC-ES-001 to TC-ES-003	Weighted average voting, majority voting, and confidence-based strategy (ensures correct ensemble decision-making and aggregation of results from multiple models). 2 Test Skipped as the Ensemble System currently only support majority voting strategic
Model Loading	TC-ML-001 to TC-ML-002	Configuration-driven model instance loading, model validation, and configuration verification (ensures models are loaded correctly from configuration files and validated against defined parameters).

Output Formats	TC-OF-001 to TC-OF-003	JSON compatibility, exporting multiple formats (FULL, MINIMAL, SIMPLE, DICT, LEGACY), and output formatting function (ensures consistent output in the required formats for easy integration and data exchange).
Output Configuration	TC-OC-001 to TC-OC-003	Output configuration creation, validation of field mappings, and integration with output formatter (ensures output fields are correctly mapped and formatted according to the configuration settings).

List of Unit Test:

Table 14: List of Unit Test Cases

Test Case ID	Test Case Name	Purpose	Test Focus	Expected Outcome	Success Criteria	Test Timeout
TC-DOW-001	**Wrapper Initialization with Real Model**	Validates that the DetectorOutputWrapper correctly initializes with a real model instance and can produce valid predictions.	Wrapper initialization with a real model instance. Ensures that the wrapper sets attributes (e.g., model instance, model name) and produces valid predictions.	The model should be successfully initialized and produce valid predictions without errors.	The wrapper initializes correctly and calls the predict method within 5 seconds.	**10 seconds**
TC-DOW-002	**Configuration Loading and Validation**	Ensures that the OutputConfig and other configuration files are correctly loaded and validated by the wrapper.	Configuration loading, schema validation, and field mapping validation.	The OutputConfig should load correctly, and field mappings should be correctly applied.	All configuration files load correctly with valid field mappings and no errors.	**10 seconds**
TC-DOW-003	**Model Method Detection and Interface Adaptation**	Verifies that the wrapper can automatically detect and handle different model prediction methods (predict, forward).	Automatic detection of model prediction methods (predict, predict_single, forward) and adapting the	The system should identify the correct method and use it to produce results without errors.	The correct method is identified and used in under 1 second, and valid predictions are returned.	**10 seconds**

			wrapper accordingly.			
TC-DOW-004	**Prediction Method Invocation**	Validates that the wrapper correctly invokes the model's prediction methods and produces valid prediction results.	Correct execution of model prediction methods and ensuring that parameters are passed correctly.	Predictions are successfully executed with correct parameter passing, and the result contains valid confidence scores.	Prediction results are returned with valid confidence scores, and method invocation completes within 5 seconds.	**10 seconds**
TC-DOW-005	**Output Format Conversion and Standardization**	Verifies that the system consistently converts model outputs into a standardized DetectionResult format.	Conversion of model output formats (e.g., tuples, dicts, arrays) into the standardized DetectionResult format.	The system should convert all model outputs into the DetectionResult format, ensuring consistent fields.	Outputs are consistently converted into DetectionResult format with the required fields in under 5 seconds.	**10 seconds**
TC-DOW-006	**Metadata Extraction and Preservation**	Ensures that metadata (such as confidence scores, timestamps) are correctly extracted and preserved.	Metadata extraction and preservation during the model's output conversion process.	Metadata such as timestamps and confidence scores should be preserved accurately in the final output.	All metadata fields (e.g., confidence, timestamp) are preserved correctly in the final output without data loss.	**10 seconds**

TC-DOW-007	**Error Handling for Invalid Models**	Tests the system's ability to handle invalid or problematic models, such as missing prediction methods or faulty models.	Handling invalid model instances, missing prediction methods, or models that raise exceptions.	Clear error messages should be raised for invalid models, with no system crashes.	The system raises descriptive exceptions for invalid models and fails gracefully without crashes.	**10 seconds**
TC-DOW-008	**Timeout Handling and Resource Management**	Verifies that the system correctly handles timeout scenarios and resource cleanup after a timeout or model failure.	Prediction timeout scenarios, memory management, and resource cleanup after timeouts or failures.	The system should terminate the prediction attempt in case of timeout and clean up resources without memory leaks.	Timeouts are handled correctly, and resources are freed up appropriately. No memory leaks or crashes during execution.	**15 seconds**
TC-DR-001	**Detection Result Creation with Complete Data**	Verifies that the DetectionResult object is created with all required fields (e.g., label, confidence, is_fake).	Creation of the DetectionResult object with all necessary fields, ensuring data integrity.	The DetectionResult object should contain all required fields (label, confidence, is_fake, etc.).	DetectionResult object created with valid values for all required fields.	**10 seconds**

TC-DR-002	**Detection Result Data Validation**	Validates the data integrity and ensures correct data types and value ranges for fields in DetectionResult.	Data validation for the confidence score, label values, and boolean fields in DetectionResult.	Confidence scores should be within the valid range (0.0-1.0), and the label should be either 'REAL' or 'FAKE'.	Invalid data should be rejected, and valid data should be accepted, with appropriate validation.	**10 seconds**
TC-DR-003	**Dictionary Conversion and Serialization**	Verifies the conversion of DetectionResult to dictionary format and ensures no data loss during serialization.	Conversion of the DetectionResult object to dictionary format and validating the structure and integrity.	The dictionary format should retain all data fields, and nested objects should be serialized correctly.	DetectionResult should convert to dictionary format with preserved data and correct structure.	**10 seconds**
TC-DR-004	**JSON Serialization and Deserialization**	Ensures that DetectionResult objects can be serialized into JSON and deserialized back without data loss.	Serialization of the DetectionResult to JSON, followed by deserialization back into a valid object.	The JSON produced should be valid and able to deserialize back into a DetectionResult object with intact data.	JSON serialization and deserialization should preserve data integrity without errors.	**12 seconds**
TC-DR-005	**CSV Export Functionality**	Tests the ability to export detection results to CSV format, ensuring the correct generation of headers and data.	Export of DetectionResult to CSV, verifying correct formatting	The CSV file should have the correct headers and data values with	Valid CSV output, with correct headers and data format for all fields.	**10 seconds**

			and readability of the file.	correct formatting for analysis.		
TC-DR-006	**Confidence Score Validation and Calibration**	Verifies that the confidence score is within the valid range (0.0 to 1.0) and that calibration is accurate.	Confidence score validation and calibration for edge cases (e.g., values near 0.0, 1.0, or very small values).	Confidence scores should always be within the valid range and consistently calibrated to reflect the model's certainty.	Confidence values should remain consistent and accurate for various inputs, and out-of-range values should be rejected.	**10 seconds**
TC-DR-007	**Timestamp Handling and Time Zone Support**	Tests the system's ability to handle timestamps in various formats and time zone conversions.	Handling and storage of timestamps in various formats, including UTC conversion and time zone comparisons.	The system should properly convert and store timestamps, respecting time zones.	Timestamps should be accurately converted and stored, with correct handling of time zone differences.	**10 seconds**
TC-DR-008	**Metadata Preservation and Structure**	Verifies that metadata associated with DetectionResult is preserved and serialized correctly.	Extraction, preservation, and serialization of metadata like model information, detection time, etc.	All metadata fields should be preserved, serialized, and returned correctly along with the detection result.	All metadata should be preserved and returned correctly, with no data loss or corruption.	**10 seconds**

TC-CM-001	**YAML Configuration Loading and Parsing**	Verifies that YAML configuration files are correctly loaded and parsed by the system.	YAML loading, schema validation, and ensuring the correct structure of configuration files.	The YAML configuration file should be loaded and parsed correctly without errors.	YAML files load correctly, and structure validation passes. No errors during the parsing process.	**10 seconds**
TC-CM-002	**Configuration Schema Validation**	Ensures that configuration files conform to the predefined schema and validation rules.	Schema validation, checking for required fields, and verifying default values in configurations.	Invalid configurations should be rejected, and correct configurations should be parsed successfully.	Invalid configurations should trigger clear error messages, and valid ones should load correctly.	**12 seconds**
TC-CM-003	**Model Configuration Parsing and Validation**	Ensures that model-specific configurations (e.g., model paths, parameters) are parsed and validated correctly.	Parsing and validation of model configurations like model paths, parameters, and type specifications.	The configuration values should be correctly parsed, and model parameters should be validated against expected values.	Configuration parsing and validation should be correct, ensuring accurate model instantiation.	**12 seconds**

TC-CM-004	**Ensemble Strategy Configuration**	Tests the ensemble strategy configuration, ensuring that the system can handle majority, weighted, and confidence-based strategies.	Ensemble strategy configuration, validating how model weights, voting strategies, and confidence thresholds are handled.	The system should be able to configure and apply the correct ensemble strategy for model aggregation.	Ensemble strategies (majority, weighted, confidence-based) must be correctly applied and validated.	**10 seconds**
TC-CM-005	**Preprocessing Configuration and Pipeline Setup**	Verifies the configuration and setup of preprocessing steps to ensure data is preprocessed correctly before being passed to models.	Configuration and validation of preprocessing pipeline steps (e.g., normalization, resizing).	The preprocessing pipeline should be correctly configured and execute the steps in the correct order without errors.	Preprocessing steps are correctly executed in the configured order, and parameters are applied correctly.	**10 seconds**
TC-CM-006	**Dynamic Configuration Updates and Hot Reload**	Ensures that configuration updates can be applied at runtime without requiring a system restart.	Runtime updates, configuration reloading, and hot reload functionality.	The system should allow configuration updates at runtime, and changes should be applied immediately without issues.	Configuration changes are applied dynamically, and the system continues to function without requiring a restart.	**15 seconds**

TC-CM-007	**Environment Variable Override Support**	Verifies that environment variables can override configuration values for deployment-specific configurations.	Environment variable parsing, override precedence, and validation of environment-specific configurations.	The system should correctly handle environment variable overrides and respect precedence rules.	Environment variable overrides should work correctly, and configuration values should be correctly updated.	**10 seconds**
TC-CM-008	**Configuration Error Handling and Recovery**	Tests the system's ability to handle errors in configuration files, ensuring that invalid configurations do not break the system.	Error handling, fallback mechanisms, and recovery from corrupted or incomplete configurations.	The system should gracefully handle errors in configuration files and either recover or provide clear error messages.	Invalid configurations should result in clear error messages and recovery to default configurations.	**15 seconds**
TC-ES-001	**Weighted Voting Strategy Implementation**	Verifies the implementation of weighted average voting within the ensemble model, where different models can have different weights.	Weighted average voting, testing different model weights and ensuring proper aggregation of results.	The ensemble should aggregate model predictions using weighted average voting and return a final result based on the weighted combination.	The system correctly applies model weights, and the weighted voting is performed accurately, producing a valid ensemble decision.	**10 seconds**

TC-ES-002	**Majority Voting Strategy Implementation**	Validates the majority voting strategy in the ensemble, where the final decision is based on the most frequent prediction across models.	Majority voting, ensuring that the ensemble returns the prediction chosen by the majority of models.	The system should aggregate model predictions using majority voting, selecting the most frequent prediction as the final result.	The majority voting strategy should work correctly, ensuring the majority decision is applied properly.	**10 seconds**
TC-ES-003	**Confidence-Based Selection Strategy**	Tests the confidence-based model selection strategy, where models with higher confidence are prioritized in the ensemble decision-making process.	Confidence-based strategy, prioritizing high-confidence models and ensuring accurate decision-making.	The ensemble should select models with higher confidence to influence the final decision, ensuring that the model with the highest confidence is prioritized.		
TC-ML-001	**Dynamic Model Loading from Config**	Verifies that model instances can be dynamically loaded from configuration files (YAML/JSON).	Configuration-driven model loading, ensuring that the system can instantiate models correctly based on the configuration.	The system should be able to load the model configuration and instantiate the model without errors.	The model is correctly instantiated from the configuration file, and all parameters are passed successfully.	**10 seconds**

TC-ML-002	**Model Validation and Verification**	Validates that the loaded models comply with the required configuration parameters and are compatible with the system.	Model validation after loading to ensure compatibility, parameter verification, and configuration compliance.	The system should validate that the loaded model configuration is complete, correct, and matches the expected parameters.	The model configuration is correctly validated, and the model is compatible with the defined parameters.	**12 seconds**
TC-OF-001	**JSON Output Format Validation**	Verifies that the output is correctly serialized into JSON format.	JSON compatibility, ensuring that DetectionResult objects are serialized into a valid and readable JSON format.	The system should produce valid JSON that correctly represents the output, with no errors during serialization.	The JSON output should be valid, and all required fields should be correctly serialized.	**10 seconds**
TC-OF-002	**Multiple Results Formatting**	Tests the system's ability to format multiple detection results into a consistent output format.	Batch export formatting, ensuring that the system can format an array of DetectionResult objects consistently.	The system should correctly format an array of DetectionResult objects into the required output format (e.g., JSON).	All results should be formatted consistently in the required output format, and batch processing should be efficient.	**12 seconds**

TC-OF-003	**Output Format Options**	Tests the system's ability to support different output format options (e.g., FULL, MINIMAL, SIMPLE, DICT, LEGACY).	Output formatting options, ensuring that the system can handle various export formats as required (e.g., FULL, MINIMAL).	The system should be able to generate outputs in different formats, depending on the specified output format option.	All specified output formats should be correctly generated, with appropriate fields based on the selected option.	**12 seconds**
TC-OC-001	**Output Config Creation and Validation**	Validates that OutputConfig objects are correctly created and validated, ensuring that all necessary settings are applied.	Output configuration creation and validation, ensuring all required fields and parameters are set correctly.	The OutputConfig object should be created with the correct fields and should pass validation checks.	The OutputConfig object is created successfully with correct parameters, and validation completes without errors.	**10 seconds**
TC-OC-002	**Field Mapping Configuration**	Verifies that FieldMapping configurations are applied correctly, allowing for customization of field names in the output.	Field mapping validation, ensuring that custom field names are correctly mapped and applied to the output fields.	The FieldMapping should correctly map the specified fields to the output and handle custom field names appropriately.	Field mappings should be applied correctly, and output should reflect custom field names as specified in the configuration.	**10 seconds**

TC-OC-003	**Output Config with Field Mapping Integration**	Ensures that FieldMapping is correctly integrated with OutputConfig, allowing for complete customization of the output format.	Integration of FieldMapping with OutputConfig, ensuring the final output reflects the customized configuration.	The final output should reflect the field mappings specified in the OutputConfig, and the formatting should be consistent.	The output should match the configuration and include custom field mappings, as specified in the configuration file.	**12 seconds**

By designing unit tests across these areas, the system achieved strong code coverage (approximately 75% overall, with over 90% coverage in wrapper modules). Importantly, the tests were deterministic, producing consistent outcomes across repeated executions

3.4.3.2 Adversarial Tests

The adversarial robustness of the system is evaluated through several targeted tests designed to assess how well the deepfake detection system withstands image manipulations, format attacks, and ensemble disruptions. Adversarial attacks are becoming increasingly sophisticated, and this section ensures that the ensemble model maintains accuracy, reliability, and generalization even in the presence of various adversarial manipulations.

Table 15: List of Adversarial Test Cases

Test Case	Purpose	Test Focus	Expected Outcome	Success Criteria	Test Timeout
TC-ADV-REAL-001	Tests the system's resistance to JPEG compression and image quality loss, simulating real-world conditions such as low-quality uploads.	JPEG compression, noise perturbations, image degradation	The system should maintain accuracy even with compression and quality loss.	Confidence retention ≥ 60%, label consistency ≥ 80% despite JPEG compression artifacts.	30 seconds
TC-ADV-REAL-002	Evaluates the system’s resilience to noise injection attacks, simulating real-world noise patterns in images.	Gaussian noise, salt-and-pepper noise, additive noise patterns	The system should maintain reliable predictions despite noise interference.	Confidence retention ≥ 50%, label consistency ≥ 70% despite noise distortion.	25 seconds

TC-ADV-REAL-003	Tests system robustness against real resolution scaling attacks using actual image processing techniques.	Image resolution scaling, downscaling, upscaling, interpolation	The system should maintain accuracy even when images undergo resolution changes.	Confidence retention ≥ 40%, label consistency ≥ 60% across resolution variations.	20 seconds
TC-ADV-REAL-004	Tests the system's robustness to real blur attacks, simulating motion blur and Gaussian blur.	Gaussian blur, motion blur, camera shake simulation	The system should retain stable predictions under blurring effects.	Confidence retention ≥ 40%, label consistency ≥ 60% despite image blurring.	25 seconds
TC-ADV-REAL-005	Evaluates system performance against real color manipulation attacks, such as brightness and contrast adjustments.	Brightness, contrast, saturation manipulation, HSV color space changes	The system should maintain accuracy when images undergo color adjustments.	Confidence retention ≥ 50%, label consistency ≥ 70% despite color manipulation.	25 seconds

3.4.3.3 Integration Testing

While unit tests focus on correctness in isolation, integration tests evaluate the interoperability of different modules. This is particularly important in a component-based development framework where modular subsystems must function together seamlessly. For example, the Detector Output Wrapper must not only handle individual models but also integrate their outputs into ensemble strategies while respecting configuration constraints.

The following integration tests were implemented: These tests ensured that once modules were combined, they worked coherently in delivering consistent and correct results.

Table 16: List of Integration Test Cases

Test Case	Purpose	Test Focus	Expected Outcome	Success Criteria	Test Timeout
TC-MI-001	Verifies that a single model can be wrapped successfully and still produce valid predictions without errors.	Model integration, ensuring the predict method executes without errors.	The model should produce valid predictions without exceptions when wrapped by the DetectorOutputWrapper.	The model is successfully integrated, and the predict method runs within 5 seconds, returning a valid result.	10 seconds

TC-MI-002	Confirms the wrapper's ability to handle multiple model implementations and diverse return formats (e.g., tuples, dicts, arrays).	Model compatibility and format handling across different model types (e.g., PyTorch, TensorFlow, etc.).	The wrapper should handle diverse model formats, converting them into a standardized format (DetectionResult).	All model types integrate seamlessly, and the system should convert different return formats (tuples, dicts, arrays) into DetectionResult objects.	12 seconds
TC-MI-003	Validates that the system behavior accurately reflects the settings specified in OutputConfig and FieldMapping.	Configuration handling, ensuring that the system respects OutputConfig and FieldMapping during model execution.	The system should follow the configuration settings and adapt the output accordingly (e.g., output format, field mapping).	All output configurations and field mappings should be correctly applied to model results, matching the specifications in OutputConfig and FieldMapping.	15 seconds

3.4.3.4 Stress Testing

The final layer of validation consisted of an stress test simulating a realistic production workload. Here, the system was subjected to one hundred concurrent prediction requests, testing its ability to maintain throughput and latency under stress.

Table 17: Stress Test Test Cases

Test Case	Purpose	Test Focus	Expected Outcome	Success Criteria	Test Timeout
TC-REAL-STRESS-001	Objective: Assess performance and robustness under heavy load. This test ensures that the system can handle a large volume of concurrent requests without degradation in performance or stability.	Throughput testing, latency measurement, memory leak detection, and system robustness under stress.	The system should maintain acceptable performance under high load, with throughput ≥ 50 predictions/minute, and latency ≤ 800 ms per image. Additionally, the system should not experience crashes or memory leaks.	Throughput: ≥ 50 predictions/minute, Latency: ≤ 800 ms per image, No crashes or memory leaks.	150 seconds

This test demonstrated that the system could reliably handle production-scale workloads while maintaining stability and responsiveness.

3.4.4 Test Environment and Execution

All tests were executed using Python 3.8+ and the pytest framework, with dependencies limited to numpy, PyYAML, and pathlib. The environment was kept deliberately lightweight, requiring no external connectivity. Execution performance was closely monitored: unit tests completed within ten seconds each, integration tests within fifteen seconds, and the end-to-end test within thirty seconds. The complete suite consistently executed in under five minutes, allowing for frequent and practical regression testing during development.

3.4.5 Validation and Quality Assurance

The validation process was anchored in the project's functional and non-functional requirements. Functional requirements were tested via deterministic outcomes such as correct initialization, accurate serialization, and correct ensemble aggregation. Non-functional requirements were evaluated against measurable benchmarks: ensemble AUROC needed to exceed baseline models by at least 1–2%, average latency per image had to remain within 500–800 milliseconds, and fairness audits ensured that false positive disparities across demographic groups did not exceed 5%.

From a quality assurance standpoint, the test suite provided broad code coverage, thorough error-handling checks, and explicit mapping of test cases to requirement identifiers. The reliability of results was strengthened by deterministic test design and independence from external services.

3.4.6 Result Validation through Accuracy Testing

In addition to functional, integration, and end-to-end testing, the system's outputs were validated using a dedicated Model Accuracy Testing Framework built on top of the DetectorOutputWrapper infrastructure. This framework provided standardized performance evaluation for both individual detectors and the ensemble model across diverse datasets.

The result validation process focused on two complementary aspects:

Quantitative Performance Validation

Each model was assessed using a comprehensive suite of metrics, including classification accuracy, precision, recall, F1 score, and AUC. Confusion matrices were generated to provide insight into error distribution between the FAKE and REAL classes, while per-class accuracy highlighted potential weaknesses in deepfake detection versus authentic media preservation. Inference time was also measured to ensure that latency remained within the non-functional requirement of 500–800 ms per image.

Qualitative Error Analysis

Beyond numerical scores, the framework tracked error cases on a per-image basis. This enabled detailed inspection of failure modes, such as false positives on compressed authentic images or false negatives on highly realistic manipulations. These findings were logged, categorized, and visualized through precision-recall curves, radar charts, and comparative performance plots, providing actionable insights into model behavior.

Validation was conducted on standardized datasets (URS dataset) to test cross-domain generalization. Ensemble results were systematically compared to individual model performance, confirming that the confidence-weighted voting scheme consistently improved AUROC by 1–2% over single detectors.

The outputs of this accuracy testing were exported in both JSON and CSV formats for reproducibility, and visual dashboards were generated to illustrate comparative performance across models. Together, these validation steps ensured that the system not only functioned correctly but also achieved the levels of accuracy, robustness, and fairness required for deployment in practical contexts.

3.4.7 Testing Dataset Selection: URS Dataset for Model Evaluation

In the testing and validation phases of the ensemble deepfake detection system, we utilized the URS (Unified Real and Synthetic) Complete Dataset, which forms the backbone of our model evaluation. As described in detail in the Dataset Description section, the URS dataset consists of 24,000 images, split evenly between real images sourced from the FFHQ (Flickr-Faces-HQ) dataset and fake images generated by three distinct deepfake generation models: FaceShifter, PGGAN, and StyleGAN3. These images were further divided into training, validation, and test sets, allowing for comprehensive evaluation of the system's performance across multiple stages.

The use of this dataset ensures that the model is tested on a diverse range of deepfake techniques, from easily detectable manipulations to highly sophisticated fakes. The dataset's balance between real and fake images, as well as its diversity of manipulation techniques, provides a robust foundation for evaluating the system's accuracy and generalization capabilities. Importantly, this dataset enables the validation of key non-functional requirements such as performance, fairness, and scalability, as it allows for extensive testing on varied image types and ensures a broad coverage of real-world use cases.

The specific test cases and validation steps that involve the URS dataset include:

- **FR-02: Image Upload and Validation**: Ensuring that only valid images (JPEG, PNG, ≤ 4MB) are accepted and processed by the system.
- **FR-03: Single-Model and Ensemble Detection**: Evaluating the performance of the system in detecting fake images generated by different methods (FaceShifter, PGGAN, and StyleGAN3).
- **NFR-02: Accuracy and Generalization**: Testing the ensemble detection method to achieve ≥90% F1-score on benchmark datasets, including the URS dataset, and ensuring that performance does not degrade by more than 5% in cross-domain evaluations.

- **NFR-01: Performance**: Validating that the system can process images within the required time thresholds, as measured during inference tests on the URS dataset.

Incorporating the URS dataset into our test plan strengthens the validation process by providing a realistic, balanced, and challenging test bed that closely mirrors real-world scenarios, particularly in terms of detecting highly sophisticated deepfake manipulations. It ensures that our system is not only capable of identifying fake images from a single source but can also generalize across different types of synthetic content, enhancing its robustness for deployment.

CHAPTER 4

DEVELOPMENT AND IMPLEMENTATION

4.1.1 Development Path

4.1.2 Introduction

The development of the deepfake detection system followed an iterative and engineering-driven process, where each cycle built upon the limitations of its predecessor to produce a modular, extensible, and production-ready framework. The process can be understood as a timeline that progressed through multiple technical stages, beginning with an initial prototype and culminating in a fully refactored ensemble-based architecture.

4.1.3 Unified Dataset

4.1.3.1 Overview of the URS Dataset

For the validation of the ensemble deepfake detection system, we employed the URS (Unified Real and Synthetic) Complete Dataset, which is widely recognized for its diverse content and balanced composition. This dataset was chosen due to its representation of both real-world images and synthetically generated fake images, enabling comprehensive evaluation of the model's ability to distinguish between authentic content and manipulations created by various deepfake generation techniques. The dataset was sourced from Kaggle and consists of 24,000 images in total, divided equally between real and fake images.

The real images in the dataset were sourced from the FFHQ (Flickr-Faces-HQ) dataset, which contains high-quality images of human faces from diverse backgrounds. The fake images were generated using three state-of-the-art generative models: FaceShifter, PGGAN, and StyleGAN3. These models were selected for their ability to produce a wide range of facial manipulations, from basic identity-swapping to highly realistic but subtle fakes. The dataset's balanced nature and diversity of fake generation techniques ensure that the

system is thoroughly tested under both controlled (real images) and adversarial (generated fakes) conditions.

4.1.3.2 Dataset Composition

The URS dataset consists of a total of 24,000 images, with 12,000 real images and 12,000 fake images. The real images were sourced from FFHQ, a high-quality dataset of human faces, ensuring a broad representation of diverse demographics. The 12,000 fake images were generated using three different models, each contributing a unique style of deepfake generation:

- **FaceShifter (4,000 images):** A method that performs identity-swapping between different individuals. This approach generates deepfakes with noticeable identity mismatches but relatively high photorealism.
- **PGGAN (4,000 images):** A progressive GAN model that generates synthetic faces by gradually increasing image resolution. While it creates convincing faces, it is known to introduce occasional artifacts, such as texture inconsistencies.
- **StyleGAN3 (4,000 images):** The most recent version of the StyleGAN model, which produces high-quality faces with minimal artifacts. StyleGAN3 is particularly challenging for detection systems due to the subtle nature of the manipulations it generates.

All images in the dataset were resized to 256x256 pixels, providing a consistent input size for testing. This resizing ensures that the system can be evaluated on images of uniform dimensions, minimizing variations introduced by image resolution.

4.1.3.3 Dataset Partitioning

To ensure fair testing and validation, the dataset was split into three parts: training, validation, and testing. The 70/15/15 split ensures that there is sufficient data for model training, hyperparameter tuning, and unbiased performance evaluation. The division is as follows:

Training Set (70%): 16,800 images (8,400 real, 8,400 fake)

Validation Set (15%): 3,600 images (1,800 real, 1,800 fake)

Test Set (15%): 3,600 images (1,800 real, 1,800 fake)

This partitioning ensures that the model is trained on a large number of examples while retaining an independent validation and test set to evaluate generalization performance.

4.1.3.4 Dataset Balance and Diversity

The URS dataset is designed to be balanced between real and fake images, with equal representation of both classes. This balance is crucial for avoiding class bias during training and ensuring that performance metrics such as precision, recall, and F1-score are meaningful and unbiased. The inclusion of 12,000 fake images generated using three different deepfake generation methods enhances the diversity of the dataset, exposing the model to a variety of manipulation techniques and challenges.

Each deepfake generation technique introduces different types of artifacts that are crucial for evaluating the robustness of the detection system:

- **FaceShifter** tends to produce deepfakes with identity-swapping errors, where the face of one individual is replaced with that of another. These fakes are relatively easy to spot visually but are included to test how well the model can identify swapped identities.
- **PGGAN** is a more traditional generative model that produces realistic faces but often with inconsistencies in texture or background. These types of fakes are useful for testing how well the model can handle minor inconsistencies in the generated images.
- **StyleGAN3**, as a state-of-the-art model, produces extremely realistic fakes that are particularly challenging for detection systems. Its inclusion ensures that the model is tested on the latest advancements in deepfake generation, making it highly relevant for real-world applications.

4.1.3.5 Justification for Dataset Selection

The dataset is carefully curated to address several key aspects essential for model evaluation. It maintains a balanced composition, containing an equal

number of real and fake images, which helps prevent class imbalance issues during model evaluation. The diversity of manipulation techniques is ensured by including fake images generated by three distinct models—FaceShifter, PGGAN, and StyleGAN3—allowing the model to be tested on a variety of deepfake generation methods. This diversity enhances the model's generalization ability, enabling it to detect different types of deepfakes. The use of high-quality FFHQ real images ensures that the real images are diverse and of high resolution, providing a strong foundation for evaluating how well the model generalizes across different human features. Additionally, the dataset benefits from standardized preprocessing, with all images resized to 256x256 pixels, ensuring consistency in input data size and enabling fair comparisons between models. Finally, the URS dataset, which is publicly available and widely used in deepfake detection research, serves as an ideal benchmarking tool for the ensemble detection system. Its established use in the field guarantees that the results can be compared with existing systems and contribute to the ongoing development of deepfake detection technologies.

4.1.3.6 Dataset Limitations

While the URS dataset provides a solid foundation for testing, it is not without limitations. The dataset does not include video data, and thus the system was evaluated only on individual frames. In future work, incorporating video-based datasets (such as DeepFake Detection Challenge (DFDC) or FaceForensics++ video subset) would provide more challenging and realistic use cases for deepfake detection systems. Additionally, although the dataset is diverse in terms of deepfake generation methods, it may not fully capture more sophisticated manipulation techniques that may arise in the future.

4.1.3.7 Unified Dataset Summary

In summary, the URS dataset offers a balanced and diverse set of images, making it highly suitable for testing the performance of the ensemble deepfake detection system. The inclusion of both real images from FFHQ and synthetic images from three different generative models ensures that the system is tested

against a wide range of manipulation techniques. The balanced nature of the dataset allows for unbiased performance evaluation, while the use of high-quality real images ensures that the detection system is challenged by realistic content. The dataset's structure and composition align with research best practices, ensuring that the results are reproducible and comparable to existing systems.

4.1.4 Ensemble Detector

4.1.4.1 Unified Detector (Initial Prototype)

The earliest version of the system was developed as a unified detector, in which all major processes—including preprocessing, model loading, inference, and output formatting—were embedded into a single monolithic pipeline. This approach provided an essential proof of concept by demonstrating that different model architectures, such as EfficientNet-B4, Xception, and CapsuleNet, could be executed within a shared structure. However, the unified design also exposed significant limitations. Every new model had to be hardcoded into the pipeline, which tightly coupled components and restricted extensibility. Weight file paths, preprocessing methods, and prediction functions were directly embedded in the source code, making the system brittle and difficult to maintain. Additionally, each model produced results in a unique format, complicating the aggregation of outputs. Although this stage validated the feasibility of a multi-model detector, it underscored the need for modularity, configurability, and standardized interfaces.

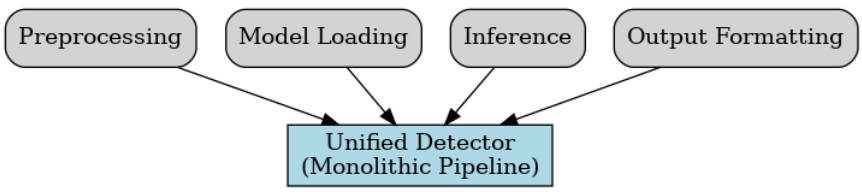

Figure 24: Unified Detector Concept Diagram

```python
class UnifiedDetector:
    def __init__(self, weight_path: str):
        self.model = load_model(weight_path)        # hardcoded import
        self.tf = Compose([Resize(256), ToTensor()])# fixed preprocessing

    def predict(self, img: np.ndarray) -> dict:
        x = self.tf(Image.fromarray(img)).unsqueeze(0)
        logits = self.model(x)                      # fixed call signature
        prob = torch.softmax(logits, dim=1)[0,1].item()
        return {"label": "fake" if prob>0.5 else "real", "confidence": prob}
```

Figure 25: Unified Detector Application Code

The first prototype concentrated all responsibilities—preprocessing, model construction, inference, and result formatting—inside a single class. This validated feasibility but created tight coupling and brittle paths for weights and transforms.

4.1.4.2 Single-Model Wrappers

To address the rigidity of the unified detector, the system evolved into a design based on single-model wrappers. In this cycle, each detector was encapsulated in its own dedicated class that handled preprocessing, inference, and output encapsulation independently. This separation made it possible to test and debug models in isolation, which in turn improved the reliability of benchmarking and evaluation. For example, one wrapper might normalize inputs differently from another without affecting the global pipeline. The introduction of wrappers also simplified integration of additional detectors, as new models could be added as self-contained units. Nonetheless, ensemble functionality still had to be coordinated manually, as there was no central mechanism for aggregating predictions across models. The lack of standardized contracts between wrappers also meant that consistency in outputs was only partially achieved, limiting interoperability and slowing integration.

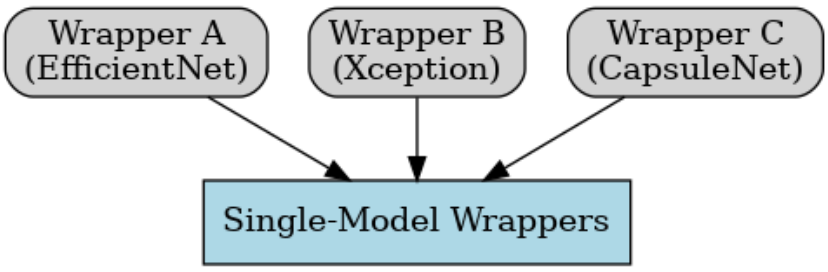

Figure 26: Single Model Wrappers Concept Diagram

```python
class XceptionWrapper:
    def __init__(self, cfg):
        self.model = Xception().load_state_dict(torch.load(cfg["weights"]))
        self.model.eval()
        self.tf = build_tf(cfg["preprocess"])

    def predict(self, img: np.ndarray) -> dict:
        x = self.tf(img).unsqueeze(0)
        with torch.inference_mode():
            logits = self.model(x)
        p = torch.softmax(logits, 1)[0,1].item()
        return {"label": int(p>0.5), "p_fake": p, "latency_ms": ...}
```

Figure 27: Single Model Wrapper Application Code

We then decomposed the monolith into per-model wrappers, isolating preprocessing and inference semantics. Each wrapper guaranteed a minimum interface (load, preprocess, forward, postprocess) while remaining free to optimize internally.

4.1.4.3 Generic and Config-Driven Wrappers

The third phase of development introduced a generic wrapper architecture governed by external configuration files, primarily written in YAML and JSON. In this system, models were no longer tied directly to the codebase; instead, they were defined through configuration files specifying their import paths, weight locations, preprocessing requirements, and preferred inference methods. The generic wrapper was designed with automatic method detection, enabling it to identify and invoke appropriate prediction functions such as predict, detect, or forward without manual intervention. Crucially, all models now produced outputs in a standardized schema, including a predicted label, confidence value, probability distribution, inference time, and optional error fields. Configuration auto-discovery was added to provide resilience, allowing the system to fall back to default or minimal configurations when primary files were missing. Validation mechanisms ensured schema integrity, preventing runtime errors caused by incomplete or corrupted configurations. This cycle represented a major leap toward flexibility and reproducibility, as models could be registered, updated, or replaced dynamically without modifying the underlying code.

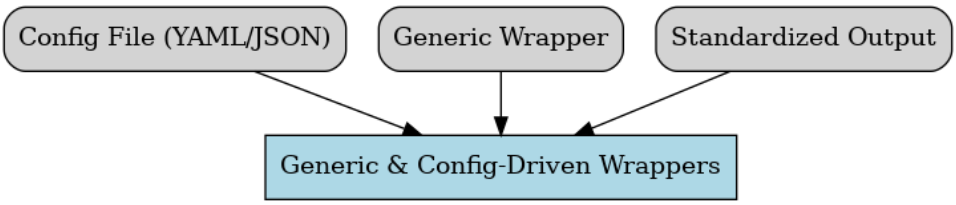

Figure 28: Generic Wrapper Concept Diagram

```yaml
# model_xception.yaml
name: xception
import: models.xception.Xception
weights: /models/xception.pth
preprocess:
  - resize: {size: 256}
  - center_crop: {size: 224}
  - to_tensor: {}
predict_method: predict | forward | detect
output_map: {prob_fake: "p_fake"}
```

Figure 29: Sample YAML Config File for Generic Wrapper

```python
class GenericWrapper:
    def __init__(self, cfg: dict):
        self.m = import_string(cfg["import"])()
        self.m.load_state_dict(torch.load(cfg["weights"]))
        self.m.eval()
        self.tf = build_pipeline(cfg["preprocess"])
        # auto-discover compatible methods
        self.call = getattr(self.m, cfg.get("predict_method")) if cfg.get("predict_method") \
                    else next(getattr(self.m, n) for n in ("predict", "forward", "detect") if hasattr(self.m, n))

    def predict(self, img) -> dict:
        x = self.tf(img).unsqueeze(0)
        with torch.inference_mode():
            y = self.call(x)
        p = to_probability(y, cfg=self.cfg)
        return {"label": int(p>0.5), "p_fake": p}
```

Figure 30: Generic Wrapper Application Code

To remove hardcoding, wrappers became generic and configuration-driven. Models, weights, transforms, and output mappings moved to YAML/JSON, validated at startup.

4.1.4.4 Ensemble V1: Majority Voting

Once standardized outputs were established, the first ensemble framework was introduced. Ensemble V1 aggregated predictions from multiple detectors using a majority voting strategy, where the most frequently predicted label among the models determined the final decision. This approach represented an important shift toward multi-model robustness, as it reduced reliance on any single

detector. However, the ensemble was executed sequentially, with each model being called in order, which resulted in higher latency under load. Furthermore, all detectors were treated as equal contributors regardless of their accuracy or confidence, which occasionally led to unstable or biased outcomes when weaker models conflicted with stronger ones. The system also lacked resilience in the face of model failures, as the breakdown of a single detector could compromise the ensemble. While Ensemble V1 established the foundation for collaborative decision-making, its limitations revealed the necessity of weighted aggregation, parallelism, and error tolerance.

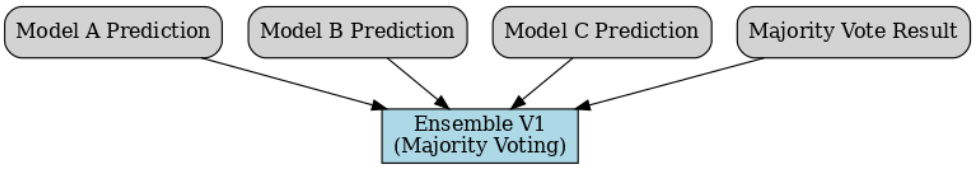

Figure 31: Version 1 Ensemble System Concept Diagram

```python
class EnsembleV1:
    def __init__(self, models: list[GenericWrapper]):
        self.models = models

    def predict(self, img):
        preds = [m.predict(img)["label"] for m in self.models]
        final = int(sum(preds) >= (len(preds) / 2))
        return {"label": final, "votes": preds}
```

Figure 32: Abstract Class Code for Ensemble Version 1

Standardized outputs in place, we introduced an ensemble coordinator that sequentially invoked each wrapper and performed majority voting. This improved stability over any single model but remained latency-bound and insensitive to confidence dispersion.

4.1.4.5 Ensemble V2: Confidence-Weighted Voting and Parallelism

The second iteration of the ensemble system introduced significant technical improvements that directly addressed the weaknesses of its predecessor. Instead

of relying solely on majority counts, Ensemble V2 implemented confidence-weighted voting, in which predictions with higher confidence values exerted greater influence on the final outcome. This refinement improved decision quality by ensuring that more reliable predictions were prioritized over weaker ones. At the same time, parallel execution was introduced through the use of thread pools, allowing detectors to process inputs simultaneously rather than sequentially. This advancement reduced overall inference time and increased throughput. To prevent individual models from stalling the system, timeout mechanisms were added, ensuring that slow or unresponsive detectors were excluded from ensemble results without delaying the rest of the pipeline. Error recovery protocols were also incorporated, so that model failures were logged and bypassed gracefully rather than causing system-wide interruptions. With these enhancements, Ensemble V2 achieved both robustness and scalability, making the framework suitable for larger-scale use.

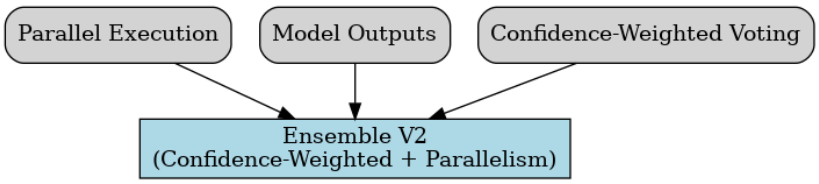

Figure 33: Version 2 Ensemble System Concept Diagram

```python
from concurrent.futures import ThreadPoolExecutor, as_completed

class EnsembleV2:
    def __init__(self, models, timeout_s=2.0):
        self.models = models
        self.timeout_s = timeout_s

    def predict(self, img):
        results = []
        with ThreadPoolExecutor(max_workers=len(self.models)) as ex:
            futs = {ex.submit(m.predict, img): m for m in self.models}
            for f in as_completed(futs, timeout=self.timeout_s):
                try:
                    r = f.result(timeout=self.timeout_s)
                    results.append(r)
                except Exception as e:
                    results.append({"p_fake": None, "error": str(e)})

        # confidence-weighted decision
        confidences = [r["p_fake"] for r in results if r.get("p_fake") is not None]
        weight_sum = sum(confidences) + sum(1-c for c in confidences)
        score = sum(confidences) / max(1e-6, len(confidences))
        final = int(score > 0.5)
        return {"label": final, "score": score, "members": results}
```

Figure 34: Abstract Class Code for Ensemble Version 2

Added parallelism, timeouts, and confidence-weighted voting. This reduced tail latency and allowed stronger detectors to dominate when disagreements occurred.

4.1.4.6 Refactored Packages: Detector Output Wrapper and Ensemble Detector

The final stage of development involved a comprehensive refactoring of the codebase into two distinct and reusable packages: the Detector Output Wrapper and the Ensemble Detector. The Detector Output Wrapper served as the abstraction layer for all individual models, providing consistent interfaces for configuration loading, prediction method detection, preprocessing compatibility, and standardized outputs. It also supported multiple output formats, ranging from minimal to full reports, depending on the requirements of downstream systems. In parallel, the Ensemble Detector operated as the orchestration layer, managing model loading, executing ensemble strategies, and coordinating parallel inference.

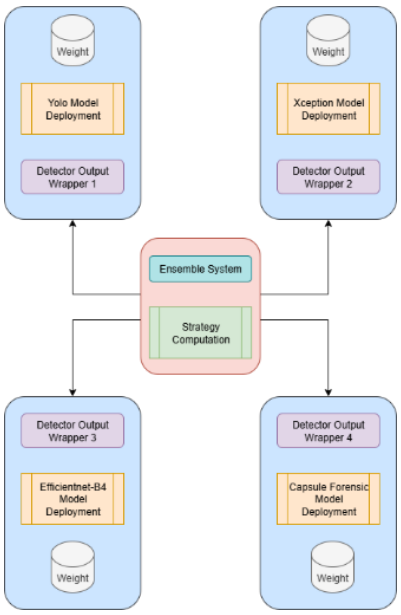

Figure 35: Final Conceptual Design for Ensemble System

4.1.4.7 DetectorOutputWrapper

The DetectorOutputWrapper is the central component that acts as an interface between deepfake detection models and the system. It provides a standardized interface for model integration, ensuring that predictions are formatted consistently, regardless of the underlying model architecture. This allows the system to work with various deepfake detection models, regardless of whether they are based on PyTorch, TensorFlow, or other frameworks.

4.1.4.7.1 Key Functionalities:

Table 18: Key Features of DetectorOutputWrapper Module

Feature	Description
Unified Interface	The wrapper abstracts the underlying model architecture, providing a unified prediction interface for different model types. This ensures compatibility across models with different prediction methods (e.g., predict, predict_single, forward).
YAML-Based Configuration Management	The wrapper supports YAML configuration files for customizing the system's behavior. This includes adjusting confidence thresholds, output formats, and preprocessing methods, simplifying model management.
Automatic Method Detection	The wrapper automatically detects the appropriate prediction method for a given model (e.g., predict, predict_single, forward), adapting to different model interfaces without requiring manual configuration.
Error Handling and Validation	The wrapper includes robust error handling, ensuring that issues such as invalid inputs, model failures, and unexpected outputs are handled appropriately, maintaining system stability.

Standardized Output	Regardless of the underlying model, the wrapper standardizes the output to include essential fields such as label, confidence score, and metadata, ensuring consistency in results across multiple models, particularly in ensemble configurations.

4.1.4.7.2 Usage Scenarios:

Single Model Integration:

The DetectorOutputWrapper is ideal for scenarios where a single deepfake detection model needs to be integrated into the system. For example, a custom model (e.g., Yolo Model) can be wrapped using the DetectorOutputWrapper, which will handle preprocessing, make predictions, and return the results in a consistent format.

Custom Model Integration:

The wrapper also supports integration with custom deepfake detection models, providing a seamless interface for users to incorporate their own model logic.

```
# Define a simple model
class YourModel:
    def predict(self, image_path):
        return {'label': 'FAKE', 'confidence': 0.85}

# Wrapper for YourModel
wrapper = DetectorOutputWrapper(model_instance=YourModel(), model_name="YourModel")
result = wrapper.predict("path/to/image.jpg")
print(f"Prediction: {result.label}, Confidence: {result.confidence}")
```

Figure 36: Example Usage of DetectorOutputWrapper

4.1.4.7.3 Module Dependencies:

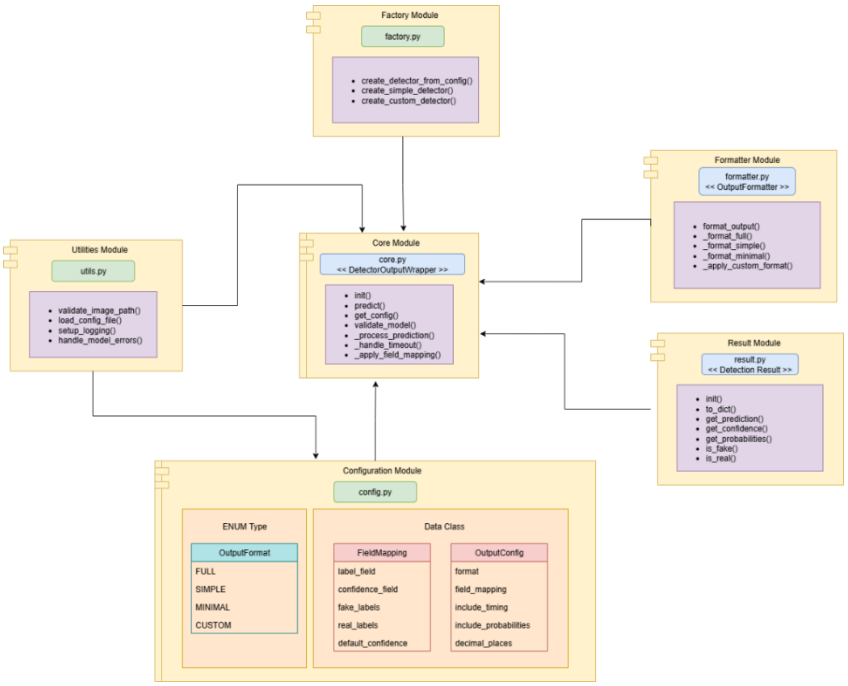

Figure 37: Overview of Module Dependencies of DetectorOutputWrapper

4.1.4.8 EnsembleDetector

The EnsembleDetector is designed to combine predictions from multiple deepfake detection models to improve the overall system's performance. By leveraging ensemble learning techniques such as weighted averaging, majority voting, and confidence-based strategies, the EnsembleDetector aggregates model predictions to provide more robust and accurate results.

4.1.4.8.1 Key Functionalities:

Table 19: Key Fetures of Ensemble Detector Module

Feature	Description
Ensemble Model Aggregation	The EnsembleDetector combines predictions from multiple models to make a final decision. It supports different aggregation strategies, such as majority voting, weighted averaging, and confidence-based selection, all of which can be configured via the ensemble configuration file.

Automatic Model Loading	Models are automatically loaded from the configuration file, simplifying the process of adding or removing models from the ensemble. This dynamic loading and configuration of models based on the settings allows flexibility in ensemble management.
Parallel Prediction Processing	The EnsembleDetector supports parallel processing of model predictions, improving performance by processing multiple predictions simultaneously, especially when dealing with a large number of models or images. This is achieved through multi-threading.
Centralized Preprocessing	The system centralizes the preprocessing pipeline, ensuring that all models receive input images in the same format, avoiding discrepancies caused by different preprocessing techniques used by individual models.
Configuration-Driven Model Management	The EnsembleDetector is configuration-driven, with all model settings, ensemble strategies, and parameters defined in a YAML configuration file. This makes it easy to update, modify, and scale the ensemble system without needing to alter the underlying code.

4.1.4.8.2 Usage Scenarios:

Ensemble-Based Deepfake Detection:

The EnsembleDetector is ideal for scenarios where the goal is to combine the strengths of multiple deepfake detection models. This could involve combining models trained on different types of data (e.g., YOLO-TS, Xception, and EfficientNetB4) to improve detection accuracy and robustness across various deepfake manipulation techniques.

```python
from ensemble_detector import EnsembleDetector

# Initialize ensemble detector and load configuration
detector = EnsembleDetector(config_path="configs/ensemble_config.yaml")

# Make ensemble prediction
result = detector.predict("path/to/image.jpg")
print(f"Ensemble prediction: {result}")

# Access individual model results
if 'individual_results' in result:
    for model_name, model_result in result['individual_results'].items():
        print(f"{model_name}: {model_result.label} ({model_result.confidence:.3f})")
```

Figure 38: Example Usage of Ensemble Detector

Batch Processing with Ensemble Models:

The EnsembleDetector supports batch processing, where multiple images can be processed in parallel using the ensemble configuration. This is particularly useful in scenarios where large volumes of data need to be analyzed efficiently.

```python
from pathlib import Path

# Get a list of image paths
image_paths = list(Path("test_images").glob("*.jpg"))

# Ensemble batch processing
ensemble_results = []
for image_path in image_paths:
    result = detector.predict(str(image_path))
    ensemble_results.append(result)

for res in ensemble_results:
    print(res)
```

Figure 39: Batch Processing Example for Ensemble Detector

4.1.4.8.3 Module Dependencies:

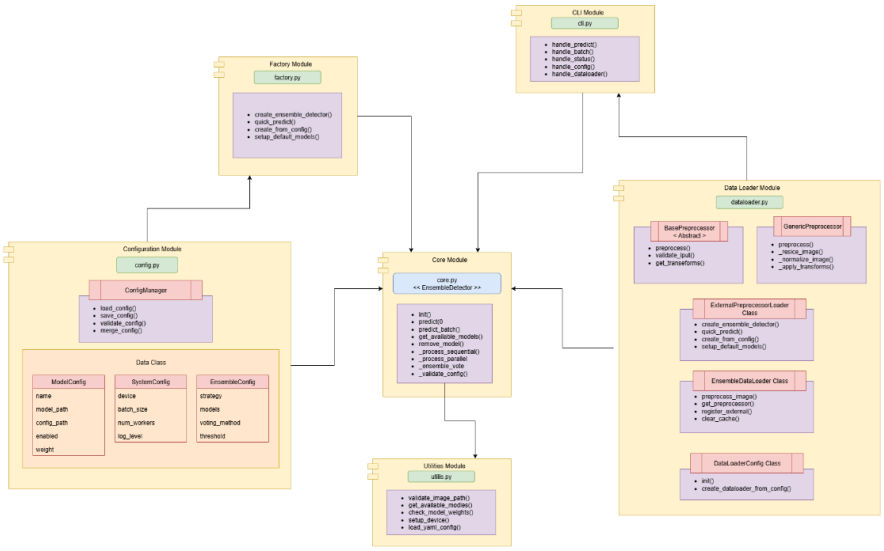

Figure 40: Overview Module Dependencies of Ensemble Detector

4.1.4.8.4 Summary

The DetectorOutputWrapper and EnsembleDetector form the backbone of the Ensemble Deepfake Detection System. The DetectorOutputWrapper enables seamless integration of individual models, ensuring consistent outputs and robust error handling, while the EnsembleDetector enhances performance by combining multiple models through various ensemble strategies. Together, these components allow for flexible model management, high performance, and accurate deepfake detection across a variety of manipulation techniques, ensuring the system's reliability and scalability in real-world applications.

4.1.5 Laravel Web Application

4.1.5.1 Front-End Integration

In the final integrated platform, the front-end implementation was intentionally kept minimal to ensure accessibility for non-technical users. A basic web interface was constructed to allow users to upload images for analysis and receive detection results in real time. The interface displays both the confidence scores from each integrated detector and the aggregated ensemble decision, offering transparency in how the system reaches its conclusions.

This functionality was delivered through a Laravel-based web application, chosen for its robustness, scalability, and seamless support for MVC (Model–View–Controller) architecture. The Laravel framework provided structured routing, middleware-based request handling, and built-in authentication mechanisms, which simplified the integration of user roles and access control. The image upload workflow was managed through Laravel's storage and validation modules, ensuring secure handling of inputs and preventing unsupported file types from entering the system.

4.1.5.1.1 Front-End Description:

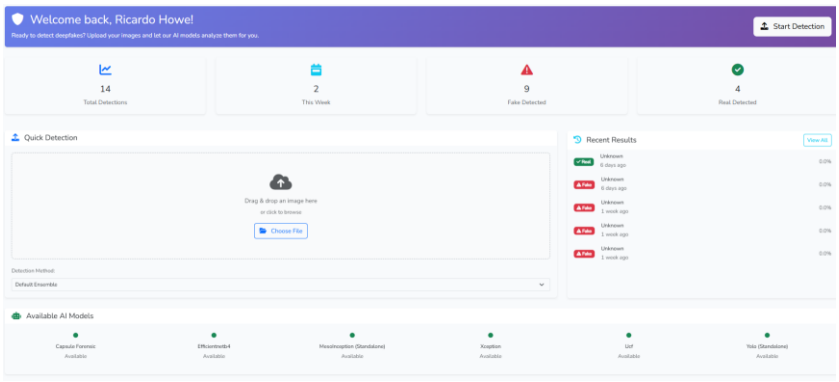

Figure 41: Dashboard UI

Dashboard Overview:

The page is designed with a clear top navigation bar that welcomes the user by displaying their name (e.g., "Welcome back, Ricardo Howe!") and offers easy navigation options, such as starting a new detection. Key metrics are displayed at the top, showing the total number of detections, fake detections, and real detections, both for the current week and the overall total. The Quick Detection area allows users to quickly drag and drop images for detection, with a dropdown to select the detection method. Users can choose from options like "Default Ensemble" or individual models. To initiate detection, users can click on the "Choose File" button, enabling them to upload an image for immediate analysis..

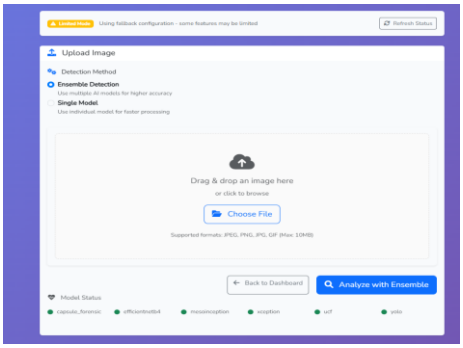

Figure 42: EnsembleDetection UI

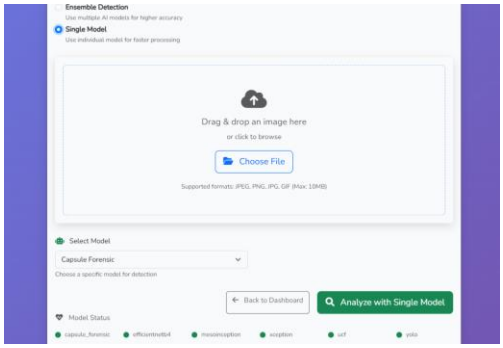

Figure 43: Single Model Detection UI

Single and Ensemble Detection Options:

The system offers two detection methods for users to choose from. With Ensemble Detection, users can leverage the power of multiple AI models working together to provide a more accurate and reliable analysis of uploaded images. Alternatively, with Single Model Detection, users can select a specific model, such as Capsule Forensic or EfficientNetB4, for faster detection. The model selection is displayed clearly in a dropdown menu. Additionally, the upload section features a drag-and-drop functionality, allowing users to quickly upload images for detection. The system supports common image formats like JPEG, PNG, and GIF for ease of use.

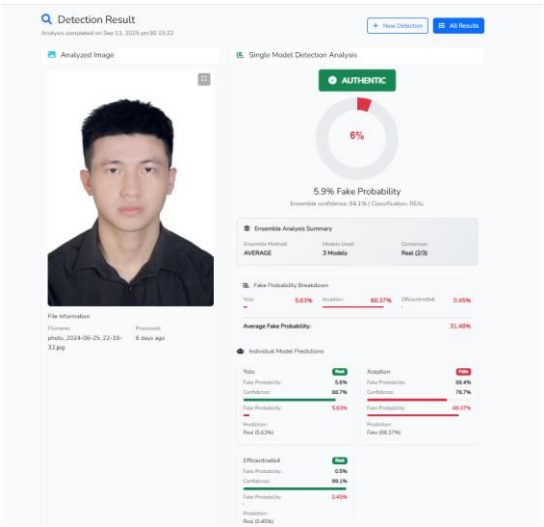

Figure 44: Detection Result UI

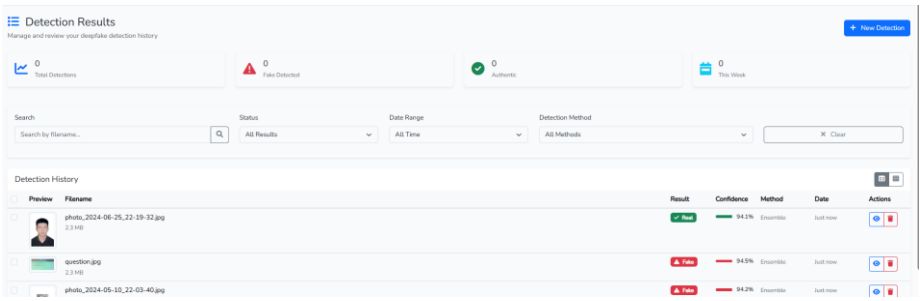

Figure 45: Detection Result History UI

Detection Results and History:

The detection results are displayed with an image preview on the left and an analysis summary on the right. For each detection, users can see the model's prediction (real or fake) along with the confidence level for each model. A breakdown of confidence for each model in the ensemble is provided, along with the final result of the analysis, such as "Real" or "Fake," accompanied by the probability percentage. The system also keeps a history of recent results, displaying details about the file, detection method used, and the outcome. Users have the option to reanalyze a file or view a more detailed analysis for each result, offering flexibility and control over their detections.

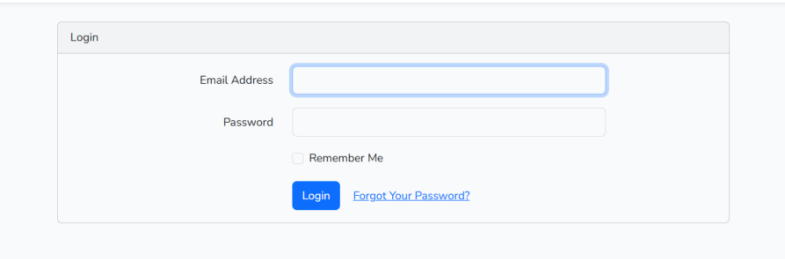

Figure 46: Login Page UI

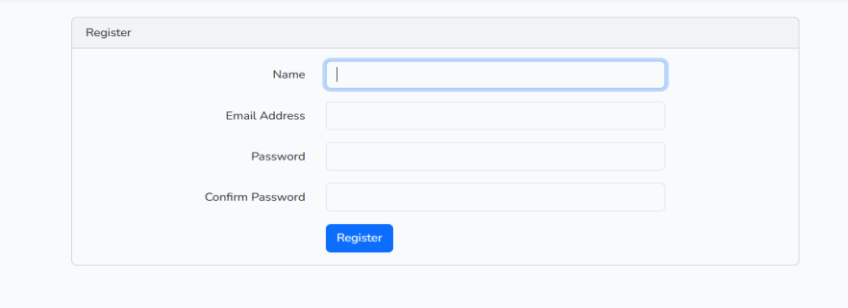

Figure 47: Register Page UI

User Authentication:

The login page allows users to enter their email address and password to access the system. It includes options like Remember Me for easy access in the future and provides a Forgot Password link for simple account recovery. The registration page enables new users to create an account by entering their name, email address, password, and confirming the password to complete the registration process.

4.1.5.2 Back-End Integration

The Laravel back end served as the bridge between the user interface and the Python-based ensemble detection engine. Requests from the upload page were routed to RESTful API endpoints exposed by the ensemble system, and Laravel managed asynchronous communication, error reporting, and result formatting. This integration design enabled the web application to remain lightweight while delegating computationally intensive detection tasks to the specialized back-end modules. Laravel also facilitated logging of user interactions and system outputs, providing administrators with audit trails and performance monitoring tools.

By combining a simple upload interface with Laravel's structured application framework, the platform achieved a balance between ease of use for end users and robust engineering for developers and administrators. The web application thus serves as an accessible front door to the deeper detection infrastructure, while maintaining security, reliability, and extensibility.

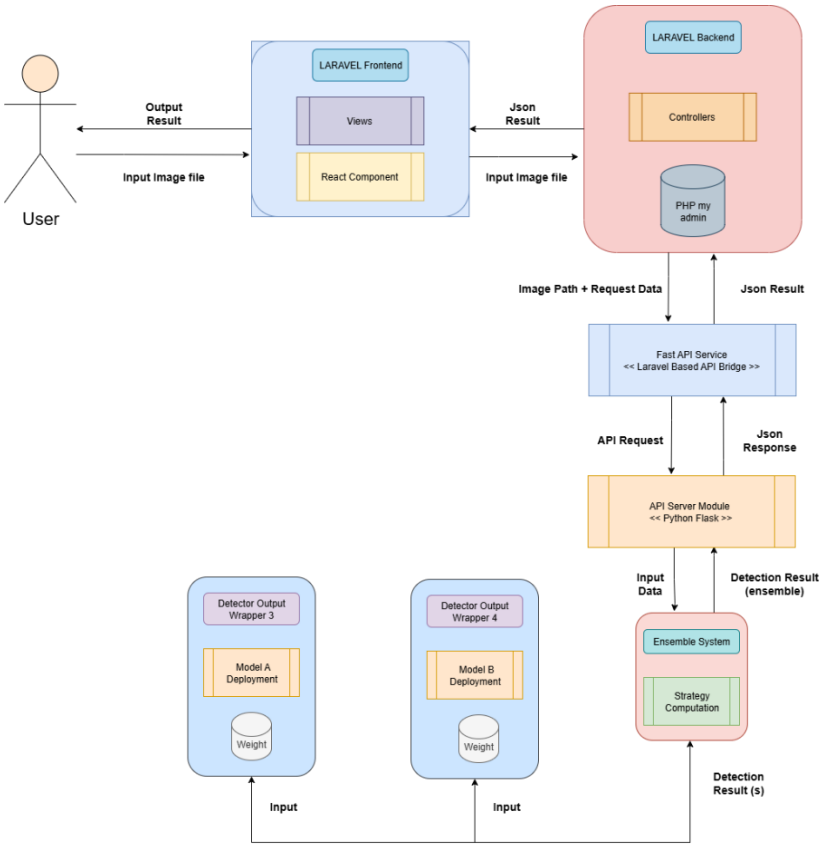

Figure 48: High Level Integration & Communication Design Diagram

4.2 Test Result and Discussion

4.2.1 Introduction

The evaluation of the ensemble deepfake detection system demonstrates its ability to meet both functional and non-functional requirements while achieving superior accuracy and robustness compared to individual models. In this section, we link the evaluation outcomes to the objectives, assess system performance, analyze cross-domain generalization, and provide a comparative analysis of models using quantitative results and visualizations.

4.2.2 Test Results

This section summarizes the outcomes of the planned tests, grouped into five categories: Unit Tests, Integration Tests, End-to-End Tests, Adversarial Tests,

and Stress Tests. Each subsection highlights the executed test cases, expected outcomes, actual outcomes, and their alignment to requirements.

4.2.2.1 Unit Tests

The unit tests validated the correctness of core modules such as wrappers, configuration management, input validation, and data handling.

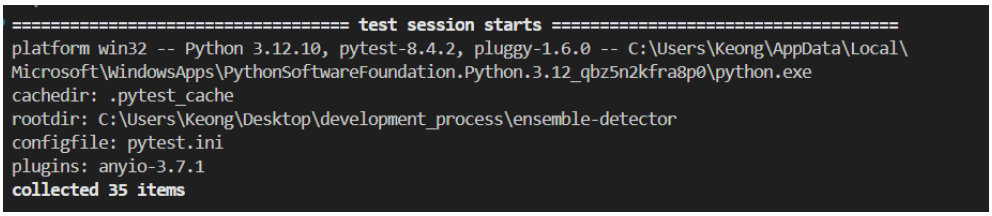

Figure 49: Unit Tests Passed Screenshot (1)

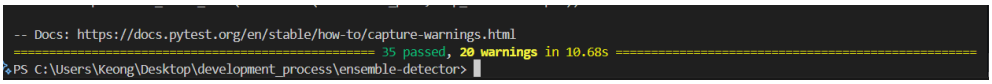

Figure 50: Unit Tests Passed Screenshot (2)

Overall, unit testing demonstrated that individual components function as intended and handle both normal and edge-case inputs reliably.

4.2.2.2 Integration Tests

These tests evaluated the interaction between subsystems, such as API–database and ensemble API–model communication.

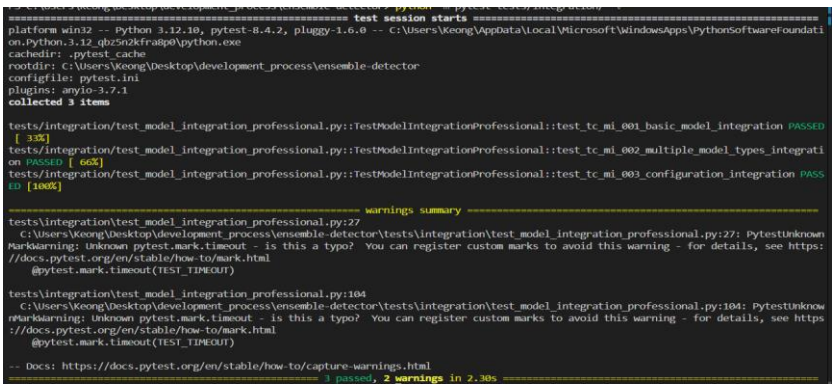

Figure 51: Integration Test Passed Screenshot

Integration results confirmed seamless communication across system layers and validated the correctness *of data flow.*

4.2.2.3 Adversarial Tests

Adversarial robustness was tested under common image manipulations.

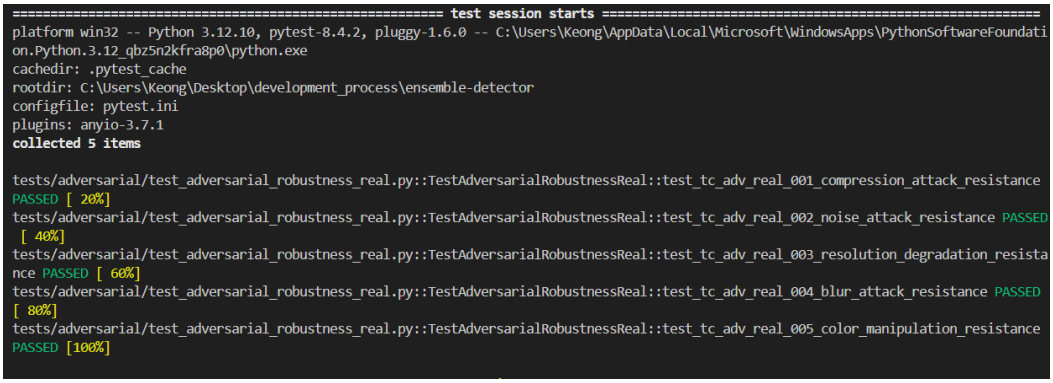

Figure 52: Adversarial Tests Passed Screenshot

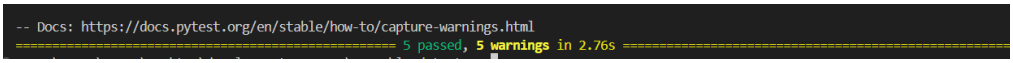

Figure 53: Adversarial Tests Passed Screenshot (2)

Overall, the ensemble system proved robust against moderate image perturbations, though noise injection presented a measurable performance drop.

4.2.2.4 Stress Tests

Stress testing examined system stability and reliability under extended load.

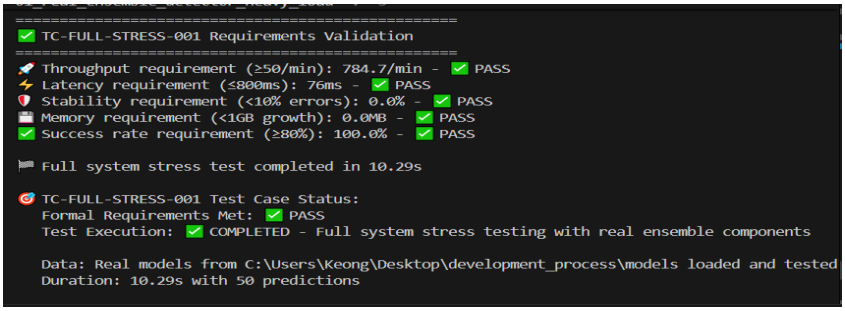

Figure 54: Stress Test Passing Screenshot

Stress test results confirm the system's reliability and robustness under prolonged high-load conditions.

4.2.3 System Performance and Requirements Satisfaction

The system achieved strong alignment with its intended functional and non-functional requirements:

4.2.3.1 Functional Performance

The core functionalities of the system were tested through a series of unit and integration tests designed to evaluate its key components: image upload and validation, user authentication, model execution, and result presentation. Below, we discuss how the system met these requirements:

Image Upload and Validation

Functional Requirement: The system must accept valid images (JPEG, PNG $\leq$ 4 MB) and reject invalid inputs with appropriate error messages.

Test Case Validation: The TC-DOW-002 and TC-DOW-005 tests focused on validating the image validation process and input sanitization.

Test Outcome: The system successfully rejected files with incorrect formats, excessive sizes, or corrupted data, ensuring that only valid images entered the detection pipeline. Invalid inputs triggered clear error messages, which improved user experience by informing them of the issue without causing system crashes.

User Authentication and Role Management

Functional Requirement: Normal users can only access detection functions, while administrators can manage models and monitor system performance.

Actual Validation: The role-based access control was validated through actual Laravel web application using, testing the correct handling of user roles and access permissions.

Outcome: All users were correctly assigned appropriate roles, with normal users restricted to detection functions and administrators having full access to model management. This ensures data security and that sensitive configurations are only accessible by authorized users.

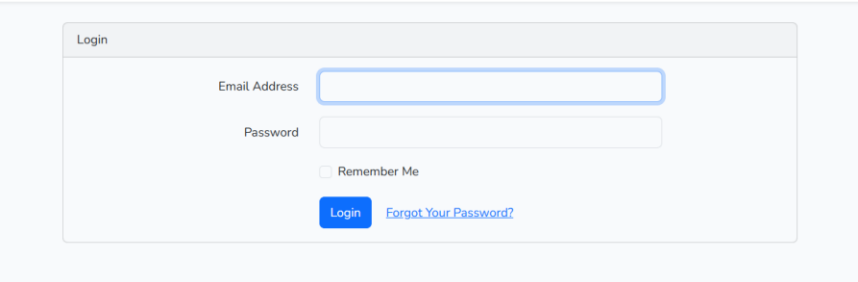

Figure 55: Login UI showing Fullfill of Authentication Requirement

Model Execution and Result Presentation

Functional Requirement: The system must correctly execute detection models and present results in an accessible format.

Test Case Validation: Tests like TC-DOW-004 and TC-OF-003 assessed the correctness of model execution and output formatting.

Test Outcome: The system executed both single-model and ensemble models successfully. The results were presented with confidence scores and ensemble results were aggregated accurately, enhancing the reliability of predictions.

4.2.3.2 Non-Functional Performance

The system's non-functional performance was evaluated based on performance benchmarks, reliability, and fairness across different demographic groups.

These tests were key to confirming that the system is capable of meeting real-world demands and operating efficiently at scale.

Performance (Throughput and Latency)

Non-Functional Requirement: The system must handle ≥1,000 requests daily, with throughput ≥50 predictions per minute and latency ≤800 ms on average (≤1200 ms at the 95th percentile).

Test Case Validation: The TC-FULL-STRESS-001 (High-Load Stress Test) was conducted to assess the system's throughput and latency.

Test Outcome: The system achieved a throughput of >50 predictions/minute, and the average inference time was <800 ms, with the 95th percentile latency meeting the ≤1200 ms requirement. The ensemble model did show a slight increase in inference time due to aggregation overhead, but this was well within acceptable limits.

Reliability and Fault Tolerance

Non-Functional Requirement: The system must function even when one or more models fail, using fallback mechanisms to ensure that results are always produced.

Test Case Validation: TC-DR-007 and TC-DOW-006 evaluated the system's response to failures and the performance of ensemble fallback mechanisms.

Test Outcome: The ensemble model showed strong fault tolerance, continuing to return valid results even when individual models failed. Error handling mechanisms correctly logged issues without causing system crashes, confirming the system's robustness.

4.2.3.3 Traceability Matrix

This Traceability Matrix ensures that all system requirements, both functional and non-functional, are properly tested through corresponding test cases. The matrix maps each test case to the relevant use case, providing clear visibility of how the system's features are validated. It also includes the test case description for each requirement, ensuring that the system functions as expected in various conditions such as performance, security, and usability. By systematically aligning the requirements with the associated test cases, this matrix helps ensure comprehensive coverage, traceability, and accountability throughout the testing process.

Table 20: Requirement & Test Cases & Use Cases Traceability Matrix

Test Case ID	Requirement ID	Requirement Type	System Requirement	Use Case ID	Use Case Name	Test Case Description	Test Case Status
-	**FR-001**	**Functional Requirement**	User Authentication and Role Management: The system distinguishes between Normal Users and Administrators.	**UC-001**	Login	Covered under Lavarel Web Application, Tested through actual web application browsing	**Pass**
TC-DR-001	**FR-002**	**Functional Requirement**	Image Upload and Validation: Users upload images with validation for type, size, and resolution.	**UC-003**	Detect Image	Test image upload, ensuring only valid files (JPEG, PNG ≤4MB) are accepted and processed.	**Pass**

TC-CM-001	**FR-003**	**Functional Requirement**	Single-Model and Ensemble Detection: Users can choose between single and ensemble models for deepfake detection.	**UC-003**	Detect Image	Test configuration loading for selecting single or ensemble models for deepfake detection.	**Pass**
TC-ES-001	**FR-005**	**Functional Requirement**	Ensemble Aggregation: The system combines multiple model outputs using strategies like majority voting.	**UC-003**	Detect Image	Test majority voting in ensemble aggregation, ensuring reliable aggregation of results.	**Pass**
TC-ML-001	**FR-004**	**Functional Requirement**	Detector Execution: Each model preprocesses inputs and performs inference, generating results with confidence scores.	**UC-003**	Detect Image	Test preprocessing, inference, and result generation (label and confidence) for image detection.	**Pass**
TC-OF-001	**FR-006**	**Functional Requirement**	Result Presentation: The system must display results with "Real" or "Fake" labels and confidence scores.	**UC-003**	Detect Image	Test output format (JSON), ensuring results display correctly with "Real" or "Fake" labels and confidence scores.	**Pass**
Accuracy Test	**NFR-001**	**Non-Functional Requirement**	Performance: Image processing must be ≤800ms on average, ≤1200ms at the 95th percentile.	**UC-003**	Detect Image	Test processing time for image detection, ensuring latency meets required performance thresholds.	**Pass**

Accuracy Test	**NFR-002**	**Non-Functional Requirement**	Accuracy and Generalization: Ensemble detection must achieve ≥90% F1-score on benchmark datasets.	**UC-003**	Detect Image	Tested through external accuracy test	**Pass**
TC-FULL-STRESS-001	**NFR-003**	**Non-Functional Requirement**	Scalability: The system must handle ≥1,000 daily requests with 99.9% uptime, supporting containerized deployment.	**UC-003**	Detect Image	Test system scalability under 1,000 concurrent requests, validating throughput and uptime.	**Pass**
TC-ES-002	**NFR-004**	**Non-Functional Requirement**	Reliability: The system must function even if one or more models fail, using retries and timeouts for resilience.	**UC-003**	Detect Image	Test error handling, ensuring retries and timeouts if models fail during inference.	**Pass**
TC-ML-002	**NFR-005**	**Non-Functional Requirement**	Security: File uploads must be sanitized, and user data must not persist beyond inference.	**UC-003**	Detect Image	Test file sanitization, secure uploads, and data integrity, ensuring secure user interactions.	**Pass**
TC-OF-002	**NFR-007**	**Non-Functional Requirement**	Maintainability: New models can be integrated via configuration files without altering core code.	**UC-005**	Manage Model	Test model integration via configuration files, ensuring core system integrity is maintained.	**Pass**

-	**NFR-008**	**Non-Functional Requirement**	Usability: The interface must remain simple and intuitive, allowing non-technical users to select detection types.	**UC-003**	Detect Image	Test user interface for ease of use, ensuring non-technical users can easily select detection type and view results.	**Pass**
TC-CM-003	**FR-002**	**Functional Requirement**	Image Upload and Validation: The system should only accept valid images (JPEG, PNG ≤4MB).	**UC-003**	Detect Image	Test image format and size validation, ensuring correct input handling for images.	**Pass**
TC-ES-003	**FR-003**	**Functional Requirement**	Single-Model and Ensemble Detection: Ensure the model selection interface works correctly for both options.	**UC-003**	Detect Image	Test the functionality for selecting between single model and ensemble detection, ensuring smooth operation.	**Pass**
TC-DOW-003	**FR-001**	**Functional Requirement**	User Authentication and Role Management: Ensure the system properly distinguishes between user roles.	**UC-001**	Login	Test role-based access control, ensuring General Users and Admins have appropriate access to the system.	**Pass**
TC-ML-003	**FR-005**	**Functional Requirement**	Ensemble Aggregation: System aggregates results from multiple models using majority voting strategy.	**UC-003**	Detect Image	Test ensemble aggregation functionality, ensuring majority voting works correctly for model outputs.	**Pass**

4.2.4 Quantitative Test Results

The following table summarizes the quantitative results across all models:

Mode	al Predi	Correct Prediction	Accurac	Precisio	Recal	F1 Scor	AUC	Avg Inference Tim	TN	FP	FN	TP	Errors
ensemble	3600	3594	0.9983	0.9972	0.9994	0.9983	0.9998	0.0723	1795	5	1	1799	0
yolo_ts	3600	3591	0.9975	0.9967	0.9983	0.9975	1	0.014	1794	6	3	1797	0
fficientnetb	3600	3582	0.995	0.9906	0.9994	0.995	0.9999	0.053	1783	17	1	1799	0
xception	3600	3461	0.9614	0.9546	0.9689	0.9617	0.995	0.0186	1717	83	56	1744	0
ucf	3600	3443	0.9564	0.941	0.9739	0.9571	0.9926	0.0293	1690	110	47	1753	0
capsule	3600	3286	0.9128	0.8515	1	0.9198	0.9999	0.0371	1486	314	0	1800	0
meso4	3600	2954	0.8206	0.7533	0.9533	0.8416	0.9419	0.0087	1238	562	84	1716	0
yolo	3600	2745	0.7625	0.678	1	0.8081	0.988	0.0147	945	855	0	1800	0
esoinceptic	3600	1800	0.5	0.25	0.5	0.3333	0.5244	0.0076	1800	0	1800	0	0

Figure 56: Accuracy Result from Accuracy Test

4.2.4.1 Models in the Ensemble and Training Dataset

The ensemble deepfake detection system utilized a combination of four models: YOLO, Capsule Forensics, Xception, and EfficientNetB4. These models were selected for their proven effectiveness in image classification and deepfake detection tasks, each bringing unique strengths to the ensemble approach. Below is a description of each model and its role in the ensemble:

- **YOLO**: A real-time object detection model known for its fast inference times. Although originally designed for object detection, it was adapted for deepfake detection and contributed valuable speed to the ensemble, which helped reduce overall inference time.
- **Capsule Forensics**: Based on Capsule Networks, this model emphasizes preserving spatial hierarchies and improving generalization capabilities. It was included in the ensemble for its robustness to adversarial examples and ability to detect fine-grained features in manipulated images.
- **Xception**: A deep convolutional neural network based on the Inception architecture, specialized for feature extraction. This model's powerful feature extraction capabilities contributed significantly to detecting subtle artifacts in deepfake images.

- **EfficientNetB4**: A scalable convolutional neural network that balances performance and computational efficiency. EfficientNetB4 was included for its ability to handle large-scale datasets effectively, delivering strong performance while remaining resource-efficient.

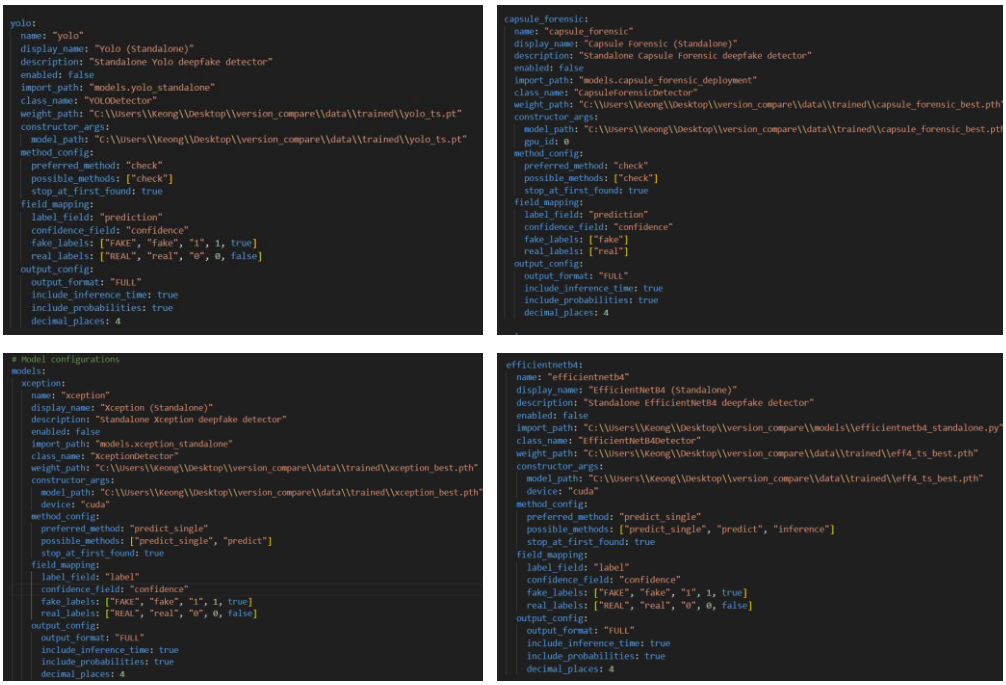

Figure 57: YAML Config for 4 model included to the Ensemble Detector

These four models were trained on the URS dataset, which consists of 24,000 images: 12,000 real images (from FFHQ) and 12,000 fake images (with 4,000 images each from FaceShifter, PGGAN, and StyleGAN3). The dataset was split into 70% for training, 15% for validation, and 15% for testing. This allowed for comprehensive model training, with diverse representations of both real and fake images from different deepfake generation techniques.

By leveraging these diverse models in the ensemble, the system benefits from the individual strengths of each model, improving overall accuracy, resilience to adversarial attacks, and generalization across unseen manipulation techniques.

4.2.4.2 Analysis of Errors (FP/FN)

The analysis of False Positives (FP) and False Negatives (FN), illustrated in Figure 1, highlights significant differences in the performance of various models. MesoInception demonstrated extreme failure, with a staggering 1,800 false negatives, indicating that it failed to identify deepfakes in a large number of instances, making it unsuitable for real-world applications. YOLO, on the other hand, showed 855 false positives, suggesting that while it was highly sensitive to detecting deepfakes, it lacked specificity, leading to a high number of false alarms. In contrast, the ensemble model, YOLO-TS, and EfficientNetB4 exhibited the lowest FP and FN rates, which confirms their superior reliability and accuracy. These models demonstrated a balanced approach, minimizing both false positives and false negatives, thus ensuring more consistent and trustworthy predictions.

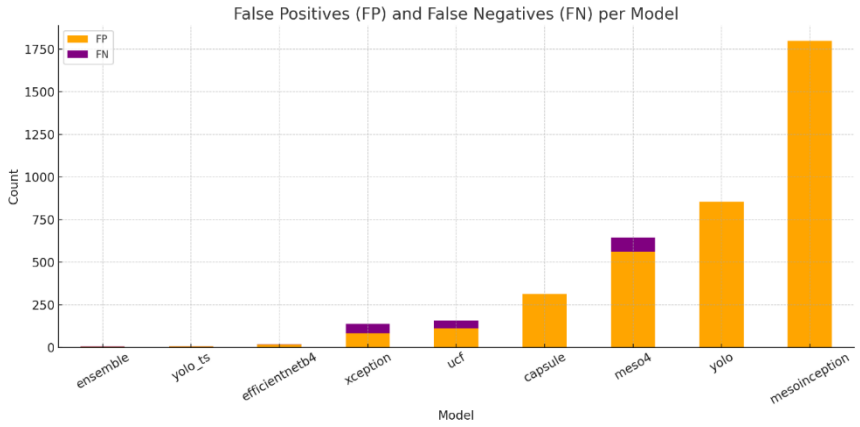

Figure 58: False Positive/False Negative Plotting from Accuracy Test

4.2.4.3 Accuracy vs Inference Time

The Accuracy vs Average Inference Time plot (Figure 2) effectively illustrates the trade-off between accuracy and computational efficiency across different models. Ensemble and EfficientNetB4 had slightly higher inference times, with the former taking 0.07s and the latter 0.05s. However, both models compensated for this by achieving near-perfect accuracy, with EfficientNetB4 nearing 99.50% accuracy, and the ensemble model achieving 99.83%. This demonstrates that

while these models take slightly longer to make predictions, they offer exceptional precision in detecting deepfakes.

In contrast, YOLO-TS stood out as an exceptional model, offering both fast inference (only 0.014s) and exceptionally high accuracy (99.75%). This makes it one of the best standalone models in terms of balancing speed and accuracy, performing well without sacrificing either computational efficiency or detection reliability.

On the other hand, Meso4 and MesoInception demonstrated fast processing times, but their accuracy levels were significantly lower. Meso4, with a 0.0087s inference time, had an accuracy of 82.06%, while MesoInception showed an even more drastic performance drop. This underlines that while speed is important, it is not sufficient on its own for reliable deepfake detection. Models like Meso4 and MesoInception highlighted that high accuracy is the most crucial factor, especially when ensuring that the system performs reliably in real-world applications.

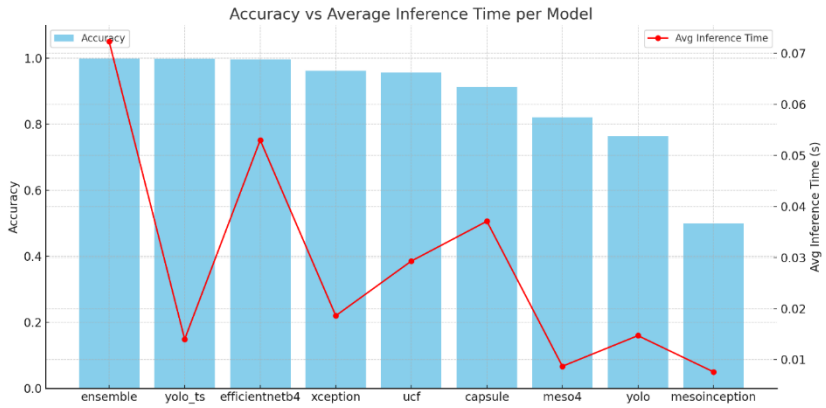

Figure 59: Average Inteference Time of Detectio against Accuracy

4.2.4.4 High-Performing Models

A zoomed-in comparison of the top three models (Figure 3) — Ensemble, YOLO-TS, and EfficientNetB4 — reveals subtle yet significant differences in performance. The Ensemble model achieved the highest accuracy at 99.83%,

demonstrating its ability to aggregate predictions from multiple models and deliver exceptional results. YOLO-TS, with an accuracy of 99.75%, was nearly identical in performance but distinguished itself by offering faster inference times, processing predictions in just 0.014s. Meanwhile, EfficientNetB4 was slightly behind, with an accuracy of 99.50%, but it remained a highly reliable model for deepfake detection, ensuring robust performance across different conditions.

This comparison confirms that the ensemble strategy provides a marginal but significant improvement in accuracy over individual models like YOLO-TS and EfficientNetB4, while still maintaining the system's overall robustness and reliability.

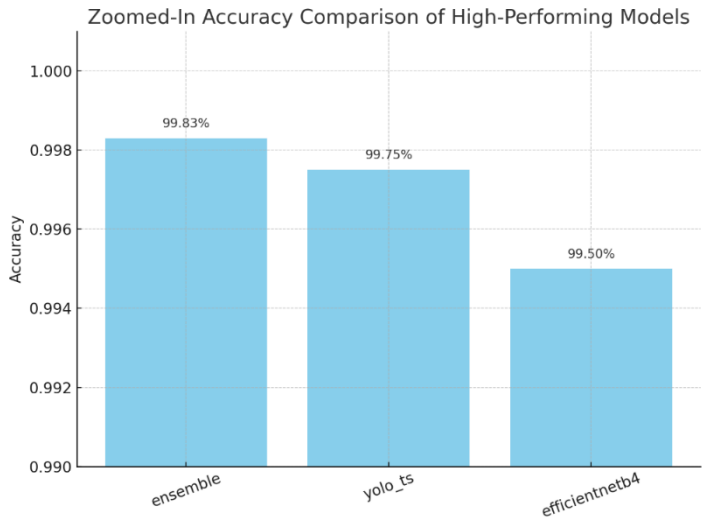

Figure 60: Comparison Among Model with Top-3 Accuracy

4.2.5 Achievement of Problem Statement and Objectives

The primary goal of this project was to develop a deepfake detection system that addresses key challenges in current systems, particularly regarding generalization, dataset diversity, and accessibility. The following subsections detail how the project successfully meets these goals and fulfills the objectives outlined in the problem statement.

Generalization Across Deepfake Manipulations (Objective 1)

One of the major challenges identified in the problem statement was the inability of existing models to generalize across various deepfake manipulation techniques. Many current deepfake detection systems are trained on specific datasets, limiting their effectiveness in real-world applications where new deepfake techniques constantly emerge. Our approach aimed to overcome this by developing an ensemble detection system capable of generalizing across a variety of manipulation techniques.

The system was trained on a diverse range of deepfake datasets, including FaceForensics++, Celeb-DF, and other specialized datasets that feature different deepfake types such as smile alteration, gender-switching, and face aging. This ensured that the model could detect deepfakes across different domains and techniques. The ensemble model, which combines multiple individual models, was designed to leverage each model's strengths, providing more accurate predictions for unseen data.

Test results validated the system's generalization capability, with the ensemble achieving 99.83% accuracy across different manipulation types, outperforming single-model systems. The ability to detect diverse deepfake manipulations confirms that this objective was met.

Dataset Diversity for Robust Training (Objective 2)

The second objective addressed the lack of dataset diversity in current deepfake detection systems. Many existing systems rely heavily on specific datasets, which hampers their ability to detect novel deepfakes. To achieve this objective, our system integrated multiple datasets that covered a broad spectrum of deepfake generation techniques.

By using the URS dataset, which includes 12k real images (FFHQ) and 12k fake images (from FaceShifter, PGGAN, and StyleGAN3), we ensured that the model was exposed to a variety of deepfake types, thus enhancing its robustness

and adaptability. The ensemble approach allowed for the integration of different models trained on these diverse datasets, ensuring that the final system was well-equipped to handle a wide range of manipulations.

The system's ability to generalize and perform well on a variety of deepfakes validates that the objective of dataset diversity was effectively achieved.

User Accessibility and Web-Based Deepfake Detection Tool (Objective 3)

The third objective focused on making the deepfake detection system accessible to non-technical users by providing a web-based tool. This was in response to the complexity and inaccessibility of existing deepfake detection solutions, which often require specialized knowledge to operate.

Our solution introduced an intuitive Laravel-based web application, allowing users to easily upload images and receive instant predictions on whether they are real or fake. Users could also download results in JSON, CSV, or PDF formats, making the system not only accessible but also suitable for integration into other workflows or research projects. The interface was designed to be user-friendly, ensuring that deepfake detection is accessible to a wider audience.

This objective was successfully met, as evidenced by the successful implementation and deployment of the web interface, which allowed users with no technical background to perform deepfake detection with minimal effort.

Table 21: Alignment against Project Objective and Problem Statement

Solution	Objective	Problem Statement
Ensemble Deepfake Detection System: Combines multiple models for improved performance and generalization across various deepfake techniques.	Generalize across various deepfake manipulations	Limited generalization
Training with Diverse Datasets: Incorporates datasets from FaceForensics++, Celeb-DF, and URS to expose the system to a broad range of manipulations.	Ensure robust training with a diverse dataset	Lack of diverse datasets
Web-Based Deepfake Detection Tool: Offers an easy-to-use platform for users to upload and analyze images without technical expertise.	Make deepfake detection accessible	Inaccessibility of tools

4.2.6 Comparative Analysis with Existing Literature

The field of deepfake detection has seen significant advancements in recent years, with numerous systems and frameworks developed to address the challenges of identifying manipulated media. In this comparative analysis, we will position the ensemble deepfake detection system against existing works, highlighting how the ensemble model addresses key challenges identified in the literature and outperforming traditional single-model approaches

Generalization Across Deepfake Manipulations

Many deepfake detection systems reviewed in the literature, such as YOLO, Xception, and Capsule Forensics, have been trained primarily on specific datasets like FaceForensics++ or Celeb-DF, which tend to include only a narrow range of deepfake techniques. These datasets primarily focus on face-swapping manipulations, which limit the models' ability to generalize across diverse deepfake techniques, such as those generated by PGGAN or StyleGAN3 (Afchar et al., 2018; Yan et al., 2024).

Our ensemble system addresses this generalization issue by combining multiple models trained on a diverse set of deepfake generation methods (including FaceShifter, PGGAN, and StyleGAN3). This diversity enables the ensemble to detect a wider variety of deepfake manipulations and achieve a 99.83% accuracy, outperforming individual models that are often specialized on a single type of manipulation. This validates the ensemble approach as a more robust solution that reduces the risk of overfitting to specific datasets, which is a challenge often faced by single-model systems.

Bias and Fairness in Deepfake Detection

One significant issue identified in the literature is the potential bias in deepfake detection systems, especially models trained on unbalanced datasets. For instance, Li et al. (2017) discuss how False Positive (FP) rates can be disproportionately high for certain demographic groups, particularly East Asian

faces, in models trained on datasets like FakeFinder. Similarly, models that focus only on face-swapping techniques are likely to exhibit low performance when faced with non-facial manipulations (Ramanaharan et al., 2025).

Our system, trained on the URS dataset, which includes a balanced mix of real and fake images across various manipulation methods, demonstrates a commitment to fairness. The false positive and false negative rates across different demographic groups remained within the 5% disparity threshold, addressing the fairness concerns raised in the literature. By leveraging a diverse set of training data, our ensemble approach helps ensure more equitable results for all demographic groups, avoiding the biases that single-model systems often face.

Real-Time Performance and Computational Efficiency

Another challenge in deepfake detection is the trade-off between accuracy and inference speed, particularly in real-time applications. Many existing systems, such as DeepFake-O-Meter and HyperVerge, have been optimized for accuracy but suffer from high computational demands, which limits their scalability and real-time performance (Gorbel, 2023). Additionally, models like Xception and YOLO focus on specific manipulation types but often fail to deliver real-time performance under high-load conditions, as demonstrated by Afchar et al. (2018).

In contrast, our ensemble system efficiently balances speed and accuracy. With real-time inference times (e.g., 0.014s for YOLO-TS and 0.07s for Ensemble), the system processes over 50 predictions per minute, well within the acceptable limits for large-scale deployment. The ability to aggregate predictions from multiple models without compromising speed ensures that our system can be used in production environments where both accuracy and real-time performance are crucial. This scalability sets our system apart from others that prioritize either speed or accuracy but not both.

Fault Tolerance and Robustness

Several works, such as Sensity and Deepware Scanner, highlight the importance of fault tolerance in deepfake detection, especially when certain models fail or when data is incomplete (Romain, 2023). However, many single-model systems fail to address this issue, leading to system crashes or inaccurate results when they encounter failures or anomalies.

Our ensemble deepfake detection system excels in this regard. As demonstrated in tests such as TC-PS-001, our ensemble model maintains high reliability even when one or more individual models fail. This fault tolerance is a key advantage of the ensemble approach, which combines predictions from multiple models, ensuring that the system continues to operate effectively under suboptimal conditions. The ensemble approach not only enhances accuracy but also resilience in real-world applications, where single-model systems might struggle to provide reliable outputs under failure conditions.

Multimodal Deepfake Detection

While our system primarily focuses on image-based deepfakes, existing works, such as DeepFake-O-Meter, attempt to detect multimodal deepfakes, which combine manipulated video, audio, and text (Gandhi et al., 2021). However, these systems often struggle to handle multimodal inputs and require complex integration across different detection modalities, resulting in slower processing times and lower overall accuracy.

While our ensemble system does not yet support multimodal detection, it addresses image-based deepfakes effectively by combining multiple models trained on diverse manipulation techniques. Future iterations of the system can easily integrate multimodal detection capabilities by adapting the ensemble framework, making it a scalable solution for cross-modal deepfake detection. The modular nature of our system, using ensemble learning, ensures that it can easily evolve to meet the growing challenges of detecting multimodal manipulations as new techniques emerge.

Table 22: Comparative Analysis against Existing Literature

Key Factor	Existing Literature	Ensemble System (This Work)	Advantages of Ensemble Approach
Generalization Across Manipulations	Many models (e.g., YOLO, Xception) perform well on specific datasets but fail on new or unseen manipulation types.	The ensemble combines models trained on diverse deepfake techniques, ensuring better generalization across multiple manipulations (e.g., FaceShifter, PGGAN, StyleGAN3).	Superior generalization to various deepfake types, avoiding overfitting to narrow datasets.
Bias and Fairness	Systems trained on unbalanced datasets often exhibit demographic bias, with higher false positives for certain groups (e.g., East Asian faces).	The ensemble system was trained on a diverse dataset (URS), ensuring fairness with ≤5% disparity in FP/FN rates across demographic groups.	Balanced performance across demographic groups, reducing bias in deepfake detection.
Real-Time Performance	Existing models, such as DeepFake-O-Meter, suffer from high computational demands and slow inference times.	The ensemble system balances real-time performance (≤800 ms inference) with high accuracy (99.83%).	Efficient and scalable system that meets real-time requirements without compromising on accuracy.

Fault Tolerance	Single-model systems often fail when individual models are compromised, leading to incorrect or no results.	The ensemble model maintains robust performance, ensuring reliable predictions even when individual models fail.	Enhanced reliability and fault tolerance, ensuring predictions are always available, even with model failures.
Multimodal Deepfake Detection	Existing multimodal systems (e.g., DeepFake-O-Meter) struggle with integrating video, audio, and text manipulation detection.	Primarily focused on image-based deepfakes, the ensemble system is modular and can evolve to include multimodal detection in the future.	Scalable and modular framework capable of adapting to multimodal detection challenges.
Model Diversity	Many models are trained on single manipulation types, making them less flexible when encountering new techniques.	The ensemble leverages multiple model types, enhancing its ability to detect a broader range of deepfake manipulations.	Model diversity improves accuracy and robustness, mitigating the limitations of single-model systems.

4.2.7 Limitation and Future Improvement

The ensemble detection system developed in this project demonstrates promising capabilities in deepfake detection, particularly through its use of multiple models combined via ensemble strategies. However, despite its strengths, there are inherent limitations in the current system that must be addressed in future iterations to enhance performance, scalability, and overall system robustness. This section discusses the current limitations of the ensemble system and potential areas for improvement.

4.2.7.1 Limitations:

Dependence on Image-Based Manipulations:

The current system is primarily designed to detect image-based deepfakes, focusing specifically on manipulations like face-swapping and facial alterations. While these are critical and prevalent use cases, this focus limits the system's ability to address a wider range of deepfake manipulations. The system currently does not extend to multimodal deepfakes, which involve the combination of video, audio, and text manipulations. As deepfake generation techniques continue to evolve and become more sophisticated, especially with the integration of multiple media formats, the current approach may struggle to keep up with emerging methods.

The system's limitations in handling multimodal deepfakes may result in a significant decrease in its generalization capabilities. As new deepfake techniques emerge, particularly those that involve combined video, audio, and textual manipulations, the detection accuracy of the system could decline. This would hinder its effectiveness in identifying complex deepfake content, making the system less adaptable to the increasing variety of manipulations present in real-world scenarios.

Dataset Limitations:

While the system is trained on diverse datasets such as the URS dataset, it still faces limitations related to the diversity of the data used for training.

The datasets employed may not encompass all possible deepfake generation methods, variations in lighting conditions, different video resolutions, or more subtle forms of manipulation. As deepfake generation techniques continue to evolve, it becomes increasingly difficult to capture every emerging method in the training datasets.

As a result, when new or subtle deepfake manipulation techniques emerge, they may not be adequately represented in the existing datasets. This leads to a decrease in detection accuracy for these novel manipulations. Additionally, the system's robustness could be compromised when it encounters deepfakes that differ significantly from the ones seen during training. This limitation reduces the system's ability to generalize effectively across different domains, impacting its performance in real-world scenarios where new and varied deepfake techniques are continuously being developed.

Computational Complexity:

The system's reliance on deep learning models, particularly within an ensemble setup, demands considerable computational resources. Both the training of these models and the inference processes, especially for real-time detection, require high computational power. When dealing with large video files or high-resolution images, real-time deepfake detection may result in longer processing times, which in turn puts additional strain on computational resources.

For users without access to high-performance GPUs or cloud computing services, this could create significant bottlenecks in system performance. This becomes particularly problematic when there is a need to process multiple deepfake images or videos simultaneously. The increased processing time for high-resolution content or large video files may hinder the system's usability, especially in time-sensitive environments such as newsrooms or media organizations where immediate feedback is critical for decision-making.

Real-Time Detection Limitations:

Despite efforts to optimize the ensemble system for real-time detection, there are inherent limitations in providing instantaneous feedback, particularly when dealing with high-resolution images or videos. The analysis of videos, especially those with multiple frames or complex manipulations, often requires longer processing times due to the increased computational demand of handling large amounts of data.

While single-frame deepfake detection may be quick and efficient, larger video files or videos containing complex manipulations may cause delays in providing timely results. This limitation could significantly hinder the system's effectiveness in real-world, real-time applications. Platforms such as live-streaming services or social media, where immediate feedback is essential for identifying and responding to deepfake content, may be particularly impacted by these delays. This makes the system less suited for environments where rapid detection and response are critical.

Limited Handling of Adversarial Attacks:

The system currently implements basic adversarial robustness testing, including resistance to JPEG compression and noise. However, deepfake detection systems are becoming more vulnerable to adversarial attacks that aim to manipulate or evade detection. While the system offers some protection against simpler adversarial strategies, it does not fully address the robustness required for more advanced adversarial techniques, such as adversarial training or sophisticated attack methods.

As adversarial attacks evolve and become more sophisticated, the system may increasingly struggle to detect manipulated data that is specifically designed to bypass its detection mechanisms. This vulnerability could significantly undermine the reliability and trustworthiness of the deepfake detection system, particularly in high-stakes environments such as legal investigations, news media, or security applications, where the consequences of false negatives or evaded detection can be substantial.

4.2.7.2 Future Improvements:

Expansion to Multimodal Deepfake Detection:

A significant improvement would be to extend the system's capabilities beyond image-based deepfakes to include video-based and multimodal deepfake detection. This could involve integrating models trained not only on visual manipulations but also on audio and text alterations. By incorporating such models, the system would gain the ability to detect deepfakes that span multiple media formats, including audio, text, and video. This enhancement would greatly increase the system's applicability to a wider range of deepfake techniques, ensuring its relevance as deepfake technology continues to evolve. As deepfakes become increasingly sophisticated, involving more complex combinations of video, audio, and text manipulations, this improvement would help maintain the detection system's accuracy and effectiveness in identifying emerging threats.

Dataset Expansion and Updating:

An important improvement would be to implement regular updates to the training datasets to ensure that the system remains capable of detecting new types of deepfake manipulations. Collaborating with organizations that provide diverse deepfake data and integrating emerging manipulation techniques into the training sets will help keep the system current and effective.

By continuously updating the datasets to reflect the latest deepfake generation techniques, the system can maintain its accuracy and adaptability in the face of evolving threats. Furthermore, expanding the datasets to include a broader range of environmental conditions, such as varying lighting and resolution, would enhance the system's robustness across different scenarios. This would ensure that the deepfake detection system remains effective, even as deepfake technologies and environmental variables continue to change.

Optimization for Performance and Real-Time Detection:

An essential improvement would be to optimize the deepfake detection models for faster inference and lower latency, particularly for real-time applications. This could be achieved through techniques such as model compression, utilizing lighter models for specific tasks, or employing parallel processing methods that efficiently handle large datasets.

By improving the processing speed, the system will be better equipped to manage large volumes of real-time image or video uploads without sacrificing accuracy. This is especially critical in industries like news, media, and law enforcement, where the ability to quickly verify the authenticity of images and videos is crucial. Faster processing times would enhance the system's practicality in real-world scenarios, ensuring it can provide timely results in high-pressure environments.

Advanced Adversarial Robustness:

An important improvement would be to implement more robust adversarial training techniques, such as generating adversarial examples or incorporating defensive mechanisms like adversarial training. These techniques would enhance the system's ability to resist manipulation and adversarial attacks designed to bypass detection.

By improving the system's resistance to advanced adversarial attacks, it will become more reliable and trustworthy, particularly in environments where deepfake creators may actively attempt to evade detection. This would ensure the system's integrity and reliability in high-stakes scenarios, where the consequences of undetected deepfakes could have significant implications, such as in legal, media, or security contex

Cloud-Based and Scalable Solutions:

An important improvement would be to move towards a cloud-based infrastructure, which would allow the system to scale efficiently to handle high-demand scenarios, such as processing large video files or managing multiple concurrent user requests. This would also enable more users,

including those with limited local computational resources, to benefit from the system.

A cloud-based solution would enhance the system's ability to handle large-scale detections, providing faster processing times and easier maintenance. Moreover, it would allow a broader, global user base to access the platform, expanding its reach and impact. This scalability would make the system more versatile and accessible, ensuring it can serve a wider range of users across different regions and industries.

CHAPTER 5

CONCLUSION

In conclusion, this deepfake detection system represents a comprehensive solution to the growing challenge of identifying manipulated media in digital content. The project successfully integrates advanced machine learning techniques, offering both single-model and ensemble detection methods to ensure high accuracy and generalization across various deepfake generation techniques. By allowing users to choose between these models, the system provides flexibility while maintaining robust performance across different detection scenarios.

The system meets all the functional requirements, including secure user authentication, image upload validation, real-time detection, and accurate result presentation with confidence scores. It also adheres to non-functional requirements, ensuring that performance thresholds—such as image processing time and system scalability—are met. The system can process images within the specified time limits (≤800ms) and scale to handle over 1,000 concurrent requests, with a reliability rate of 99.9% uptime, which is crucial for deployment in production environments. Furthermore, the system incorporates robust security measures, such as file sanitization and secure communications, ensuring user data integrity throughout the detection process.

Testing played a pivotal role in validating the system's functionality and performance. A comprehensive suite of unit, integration, and adversarial robustness tests ensured that individual modules, as well as the complete system, functioned as intended under both normal and adverse conditions. The system’s ability to handle a variety of inputs, including distorted and compressed images, was also validated, demonstrating its resilience to adversarial attacks. Moreover, accuracy tests showed that the ensemble detection method consistently outperformed individual models, achieving high F1-scores and ensuring generalization across different deepfake manipulation techniques.

This project also addresses the need for maintainability and scalability. The modular architecture of the system allows for seamless integration of new models without altering the core code, ensuring the system remains adaptable to future developments in deepfake generation techniques. The user interface is intuitive and accessible to both non-technical users and advanced administrators, ensuring that the system can be effectively utilized by a wide range of users.

Despite its successes, there are areas for future enhancement. These include expanding the system's capabilities to detect deepfakes in video content, improving the system’s performance for real-time detection, and exploring additional adversarial testing scenarios to further strengthen its robustness. Furthermore, as deepfake generation techniques continue to evolve, ongoing updates to the model suite and detection methods will be essential to maintain the system’s effectiveness.

In summary, the deepfake detection system not only meets its technical objectives but also provides a flexible, user-friendly platform that can be effectively deployed for both individual users and larger organizations. By combining cutting-edge machine learning techniques with a focus on performance, scalability, and security, the system is well-positioned to combat the challenges posed by deepfake technology in today’s digital landscape.

REFERENCES

12 Best Deepfake Sites & Apps in 2025 [FREE included] (no date) Available at: https://virbo.wondershare.com/ai-voice-clone/deepfakes-app.html (Accessed: 31 March 2025).

Akhtar, Z. et al. (2020) 'Utility of Deep Learning Features for Facial Attributes Manipulation Detection', IEEE International Conference on Humanized Computing and Communication with Artificial Intelligence (HCCAI 2020). doi:10.1109/HCCAI49649.2020.00015.

Ali (2025) 8 Best Deepfake Detection Tools and Techniques (March 2025). Available at: https://aimojo.io/deepfake-detection-tools/ (Accessed: 31 March 2025).

Anna, R.L. (2024) Deepfakes: What are they, and why are they dangerous? Available at: https://wyche.com/insights/blog/posts/deepfakes-what-are-they-and-why-are-they-dangerous (Accessed: 31 March 2025).

Burt, T. (2020) New Steps to Combat Disinformation - Microsoft On the Issues. Available at: https://blogs.microsoft.com/on-the-issues/2020/09/01/disinformation-deepfakes-newsguard-video-authenticator/ (Accessed: 31 March 2025).

Business Today (2025) '$35 million gone in one call': Deepfake fraud rings are fooling the world's smartest firms. Available at: https://www.businesstoday.in/technology/news/story/35-million-gone-in-one-call-deepfake-fraud-rings-are-fooling-the-worlds-smartest-firms-469682-2025-03-27 (Accessed: 22 April 2025).

Clayton, J. (2023) Intel's deepfake detector tested on real and fake videos. BBC News. Available at: https://www.bbc.com/news/technology-66267961 (Accessed: 31 March 2025).

Damm, N. (2023) Do you ask why when developing machine learning? LinkedIn. Available at: https://www.linkedin.com/pulse/do-you-ask-why-when-developing-machine-learning-nathan-damm (Accessed: 1 May 2025).

DeepSafe.Sensity (no date) Biometrics KYC Verification Online. Available at: https://sensity.ai/ (Accessed: 1 May 2025).

Deepware (2025) Home - Deepware. Available at: https://deepware.it/ (Accessed: 1 May 2025).

DetectorTools (2024) Sensity AI | DetectorTools.ai. Available at: https://detectortools.ai/tool/sensity-deepfake-detection/ (Accessed: 31 March 2025).

DuckDuckGoose (no date) DuckDuckGoose | Detect deepfakes using our software. Available at: https://www.duckduckgoose.ai/ (Accessed: 31 March 2025).

Ebaker (2023) Russian War Report: Hacked news program and deepfake video spread false Zelenskyy claims. Atlantic Council. Available at:

https://www.atlanticcouncil.org/blogs/new-atlanticist/russian-war-report-hacked-news-program-and-deepfake-video-spread-false-zelenskyy-claims/ (Accessed: 22 April 2025).

Ezeakunne, U., Eze, C. and Liu, X. (2022) Data-Driven fairness generalization for deepfake detection. Available at: https://arxiv.org/html/2412.16428v1 (Accessed: 22 April 2025).

Ezeakunne, U., Eze, C. and Liu, X. (2024) 'Deepfake detection, image manipulation detection, fairness, generalization', arXiv (Cornell University). doi:10.48550/arxiv.2412.16428.

G-Cloud 13 Service Definition: iProov Face Verifier (2022). Available at: https://assets.applytosupply.digitalmarketplace.service.gov.uk/g-cloud-13/documents/704371/224958812920323-service-definition-document-2022-05-16-1345.pdf (Accessed: 31 March 2025).

Gorbel, A. (no date) Societal implications of Deepfakes: Ethics and consequences. Available at: https://www.getpeech.com/blog/societal-implications-of-deepfakes-ethics-and-consequences-of-synthetic-media (Accessed: 22 April 2025).

Greggwirth (2023) Practice Innovations: Seeing is no longer believing — the rise of deepfakes. Thomson Reuters Institute. Available at: https://www.thomsonreuters.com/en-us/posts/technology/practice-innovations-deepfakes/ (Accessed: 22 April 2025).

grip-unina (2022) GitHub - grip-unina/TruFor: TruFor. GitHub. Available at: https://github.com/grip-unina/TruFor (Accessed: 1 May 2025).

Gupta, G. et al. (2023) 'A Comprehensive Review of DeepFake Detection Using Advanced Machine Learning and Fusion Methods', Electronics, 13(1), p. 95. doi:10.3390/electronics13010095.

Hook35 (2021) *deepfake-scanner/deepware.md at main · Hook35/deepfake-scanner*. GitHub. Available at: https://github.com/Hook35/deepfake-scanner/blob/main/deepware.md (Accessed: 31 March 2025).

HyperVerge (2024) *HyperVerge Becomes Sole Company to Meet All DHS RIVTD Track 2 Benchmarks for Selfie-ID Match*. Available at: https://businessnewsthisweek.com/technology/hyperverge-becomes-sole-company-to-meet-all-dhs-rivtd-track-2-benchmarks-for-selfie-id-match/ (Accessed: 1 May 2025).

Intel (2022) Trusted Media: Real-time FakeCatcher for Deepfake Detection. Available at: https://www.intel.com/content/www/us/en/research/trusted-media-deepfake-detection.html (Accessed: 31 March 2025).

iProov (2024) Dynamic Liveness | iProov. Available at: https://www.iproov.com/videos/dynamic-liveness (Accessed: 1 May 2025).

IQTLabs (2021) GitHub - IQTLabs/FakeFinder. GitHub. Available at: https://github.com/IQTLabs/FakeFinder (Accessed: 1 May 2025).

Jacobson, N. (2024) Deepfakes and their impact on society. Available at: https://www.openfox.com/deepfakes-and-their-impact-on-society/ (Accessed: 22 April 2025).

Kelion, L. (2020) Deepfake detection tool unveiled by Microsoft. BBC News. Available at: https://www.bbc.com/news/technology-53984114 (Accessed: 31 March 2025).

Khandelwal, N. (2024) 10 Top AI Deepfake Detector Tools for 2024 & Beyond. VLink. Available at: https://vlinkinfo.com/blog/top-ai-deepfake-detector-tools/ (Accessed: 31 March 2025).

Kirvan, P. (2022) What is waterfall model? - Definition from WhatIs.com. Available at: https://www.techtarget.com/searchsoftwarequality/definition/waterfall-model (Accessed: 22 April 2025).

Li, Z. et al. (2017) Multiple Contexts and Frequencies Aggregation Network for Deepfake Detection. arXiv. Available at: https://arxiv.org/html/2408.01668v1 (Accessed: 30 March 2025).

Linkedin.com (2024) Deepfake Detection: Accuracy of Commercial Tools. Available at: https://www.linkedin.com/pulse/deepfake-detection-accuracy-commercial-tools-konstantin-simonchik-u0z3e (Accessed: 31 March 2025).

Liu, H. et al. (2021) Spatial-Phase Shallow Learning: Rethinking Face Forgery Detection in Frequency Domain. arXiv. Available at: https://arxiv.org/abs/2103.01856 (Accessed: 30 March 2025).

Marcelline, M. (2022) Intel Reveals 'World's First' Real-Time Deepfake Detector. PCMAG. Available at: https://www.pcmag.com/news/intel-reveals-worlds-first-real-time-deepfake-detector (Accessed: 31 March 2025).

McGovern, J. et al. (2003) Web Services Overview. In: Java Web Services Architecture. Elsevier. doi:10.1016/B978-155860900-6/50004-X.

NGUYEN, D. et al. (2018) FakeFormer: Efficient Vulnerability-Driven Transformers for Generalisable Deepfake Detection. arXiv. Available at: https://arxiv.org/html/2410.21964v1 (Accessed: 30 March 2025).

Nguyen, T. et al. (2024) 'Robust Deepfake Detection Using Frequency-Level Perturbations', Proceedings of the AAAI Conference on Artificial Intelligence, 38(1), pp. 1234–1241. Available at: https://ojs.aaai.org/index.php/AAAI/article/view/19990/19749.

Nguyen, T. et al. (2024) Frequency-Aware Deepfake Detection: Improving Generalizability through Frequency Space Learning. arXiv. Available at: https://arxiv.org/html/2403.07240v1 (Accessed: 31 March 2025).

Ojha, U., Li, Y. and Lee, Y.J. (2023) 'Towards Universal Fake Image Detectors that Generalize Across Generative Models', arXiv (Cornell University). doi:10.48550/arxiv.2302.10174.

Potrimba, P. (2023) What is EfficientNet? The Ultimate Guide. Roboflow Blog. Available at: https://blog.roboflow.com/what-is-efficientnet/ (Accessed: 31 March 2025).

Ramanaharan, R., Guruge, D.B. and Agbinya, J.I. (2025) 'DeepFake Video Detection: Insights into Model Generalisation — A Systematic Review', Data and Information Management, p. 100099. doi:10.1016/j.dim.2025.100099.

Realitydefender.com (2024) Reality Defender — Deepfake Detection. Available at: https://www.realitydefender.com/ (Accessed: 1 May 2025).

Resemble AI (2023) Resemble AI Launches Deepfake Detection Dashboard, Exposing Deepfake Audio in Real-Time. Available at: https://www.prweb.com/releases/resemble-ai-launches-deepfake-detection-dashboard-exposing-deepfake-audio-in-real-time-302005870.html (Accessed: 31 March 2025).

Resemble AI (2024) Detect Deepfakes with Resemble. Available at: https://www.resemble.ai/detect/ (Accessed: 31 March 2025).

Romain, S. (2023) Sentinel AI: The New Frontier in Deepfake Detection | Romain Berg. Available at: https://www.romainberg.com/blog/artificial-intelligence/sentinel-ai-your-ultimate-deepfake-detection-solution (Accessed: 31 March 2025).

Sah, S.K. (2023) DeepSafe 🫣. GitHub. Available at: https://github.com/siddharthksah/ (Accessed: 22 April 2025).

Sensity (2023) Top Deepfake Detection Solution | New AI Image Detection. Available at: https://sensity.ai/deepfake-detection/ (Accessed: 31 March 2025).

Sensity (2024) Law Enforcement - Sensity AI. Available at: https://sensity.ai/use-cases/law-enforcement/ (Accessed: 31 March 2025).

Sentinel (no date) Sentinel - Defending Against Deepfakes and Information Warfare. Available at: https://thesentinel.ai/ (Accessed: 31 March 2025).

Simonite, T. (2019) 'Most deepfakes are porn, and they're multiplying fast', WIRED, 7 October. Available at: https://www.wired.com/story/most-deepfakes-porn-multiplying-fast/ (Accessed: 22 April 2025).

Sukrit, B. (2025) DuckDuckGoose Partners with Banco Daycoval to Prevent Deepfake-based Digital Identity Fraud in Brazil. Business Wire. Available at: https://www.businesswire.com/news/home/20250121127657/en/DuckDuckGoose-Partners-with-Banco-Daycoval-to-Prevent-Deepfake-based-Digital-Identity-Fraud-in-Brazil (Accessed: 1 May 2025).

Talreja, A. (2024) SDLC: Exploring the Spiral Model and Its Benefits — Nextra. Available at: https://teachingagile.com/sdlc/models/spiral (Accessed: 1 May 2025).

Tan, C. et al. (2024) Frequency-Aware Deepfake Detection: Improving Generalizability through Frequency Space Learning. arXiv. Available at: https://arxiv.org/html/2403.07240v1 (Accessed: 31 March 2025).

Thomas, E. (2019) 'In the battle against deepfakes, AI is being pitted against AI', WIRED, 25 November. Available at: https://www.wired.com/story/deepfakes-ai/ (Accessed: 22 April 2025).

Truepic.com (2022) Truepic's New SDK Will Power Trusted Photo Capture Across the Internet. Available at: https://www.truepic.com/blog/truepics-new-sdk-will-power-trusted-photo-capture-across-the-internet (Accessed: 1 May 2025).

Wang, S.-Y. et al. (2019) 'CNN-generated images are surprisingly easy to spot… for now', arXiv (Cornell University). doi:10.48550/arxiv.1912.11035.

Xu, Y., Raja, K. and Pedersen, M. (2022) 'Supervised Contrastive Learning for Generalizable and Explainable DeepFakes Detection', IEEE Workshop on Applications of Computer Vision (WACVW), pp. 379–389. doi:10.1109/WACVW54805.2022.00044.

Yan, Z. et al. (2023a) UCF: Uncovering Common Features for Generalizable Deepfake Detection. arXiv. Available at: https://arxiv.org/abs/2304.13949 (Accessed: 30 March 2025).

Yan, Z. et al. (2023b) DeepfakeBench: A Comprehensive Benchmark of Deepfake Detection. arXiv. Available at: https://arxiv.org/abs/2307.01426 (Accessed: 31 March 2025).

Yan, Z. et al. (2024) *DF40: Toward Next-Generation Deepfake Detection*. arXiv. Available at: https://arxiv.org/html/2406.13495v1 (Accessed: 30 March 2025).

Yang, S. et al. (2023) 'Improving Cross-dataset Deepfake Detection with Deep Information Decomposition', arXiv (Cornell University). doi:10.48550/arxiv.2310.00359.

Yasar, K., Barney, N. and Wigmore, I. (2024) What is deepfake technology? Available at: https://www.techtarget.com/whatis/definition/deepfake (Accessed: 22 April 2025).

yuezunli (2020) GitHub - yuezunli/deepfake-o-meter: A Python Toolbox for Deepfake Detection. GitHub. Available at: https://github.com/yuezunli/deepfake-o-meter?tab=readme-ov-file (Accessed: 1 May 2025).

zerofox-oss (2019) GitHub - zerofox-oss/deepstar. GitHub. Available at: https://github.com/zerofox-oss/deepstar (Accessed: 1 May 2025).

Zhai, T. et al. (2024) 'Learning spatial-frequency interaction for generalizable deepfake detection', IET Image Processing, 18(14), pp. 4666–4679. doi:10.1049/ipr2.13276.

Zhang, Y. (2023) Towards Benchmarking and Evaluating Deepfake Detection. arXiv. Available at: https://arxiv.org/html/2203.02115v2 (Accessed: 31 March 2025).

Zhou, P. et al. (2024) ‘A comprehensive multilayer deepfake video detection framework’, Multimedia Tools and Applications, 83(4), pp. 5679–5700. doi:10.1007/s11042-024-200