

## **ABSTRACT**

The growing prevalence of online hate speech poses significant challenges for social media platforms and online communities. Traditional hate speech detection systems, relying heavily on keyword-based filtering and manual moderation, often fail to capture the nuances of language, such as sarcasm, irony, and context. This project proposes an advanced, multimodal approach for hate speech detection, integrating text, audio, and video analysis to enhance accuracy and context awareness. By leveraging state-of-the-art machine learning models like BERT, coupled with traditional models such as Naive Bayes and SVM, the system can effectively identify hate speech across various media formats and languages. Additionally, the system incorporates multilingual support, enabling it to function on global platforms where users communicate in diverse languages. Real-time processing is facilitated via a user-friendly Streamlit interface, allowing users to upload and analyze content efficiently. This solution aims to address the limitations of existing systems by improving detection accuracy, reducing false positives and negatives, and providing a scalable, adaptable solution for online content moderation. Future enhancements include improving context-aware detection, expanding data sources, and enhancing the system's ability to understand cultural nuances.

## **TABLE OF CONTENTS**

<b>Chapter</b>	<b>Title</b>	<b>Page Number</b>
	<b>ABSTRACT</b>	
	<b>LIST OF FIGURE</b>	
	<b>LIST OF ABBREVIATION</b>	
<b>CHAPTER 1</b>	<b>INTRODUCTION</b>	
1.1	BACKGROUND OF HATE SPEECH DETECTION	
1.2	IMPORTANCE OF MULTILINGUAL DETECTION	
1.3	PROJECT OBJECTIVES	
1.4	PROBLEM STATEMENT	
1.5	SCOPE OF THE PROJECT	
1.6	METHODOLOGY	
1.7	PROJECT CONTRIBUTIONS	
<b>CHAPTER 2</b>	<b>LITERATURE SURVEY</b>	
<b>CHAPTER 3</b>	<b>SYSTEM ANALYSIS</b>	
3.1	EXISTING SYSTEM	
3.1.1	DISADVANTAGES	
3.2	PROPOSED SYSTEM	
3.2.1	ADVANTAGES	
3.3	ARCHITECTURE DIAGRAM	
<b>CHAPTER 4</b>	<b>SYSTEM REQUIREMENTS</b>	
4.1	HARDWARE REQUIREMENTS	
4.2	SOFTWARE REQUIREMENTS	
4.3	SOFTWARE SPECIFICATION	
4.3.1	FRONT-END	
<b>CHAPTER 5</b>	<b>MODULES DESCRIPTION</b>	

5.1	USER AUTHENTICATION MODULE
5.2	HATE SPEECH DETECTION MODULE
5.3	MULTIMODAL INTEGRATION MODULE
5.4	REAL-TIME PREDICTION MODULE
5.5	FEEDBACK AND REPORTING MODULE
5.6	ADMIN CONTROL AND MANAGEMENT MODULE
<b>CHAPTER 6</b>	<b>SYSTEM DESIGN</b>
6.1	UML DIAGRAMS
6.1.1	USE CASE DIAGRAM
6.1.2	SEQUENCE DIAGRAM
6.1.3	ACTIVITY DIAGRAM
6.1.4	CLASS DIAGRAM
<b>CHAPTER 7</b>	<b>TESTING</b>
7.1	TYPES OF TESTING
7.1.1	UNIT TESTING
7.1.2	INTEGRATION TESTING
7.1.3	SYSTEM TESTING
7.1.4	WHITE-BOX TESTING
7.1.5	BLACK-BOX TESTING
7.1.6	ACCEPTANCE TESTING
<b>CHAPTER 8</b>	<b>CONCLUSION AND FUTURE ENHANCEMENTS</b>
8.1	CONCLUSION
8.2	FUTURE ENHANCEMENTS
<b>APPENDIX A</b>	<b>SOURCE CODE</b>
<b>APPENDIX B</b>	<b>SCREENSHOTS</b>
	<b>REFERENCES</b>

## LIST OF FIGURES

FIGURE NO	FIGURE TITLE	PAGE NO
3.1	System Architecture	
6.1	Use Case Diagram	
6.2	Sequence Diagram	
6.3	Activity Diagram	
6.4	Class Diagram	

## LIST OF ABBREVIATIONS

Abbreviation	Full Form
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
GPU	Graphics Processing Unit
JSON	JavaScript Object Notation
LLM	Large Language Model
NLP	Natural Language Processing
REST	Representational State Transfer
SVM	Support Vector Machine
UAT	User Acceptance Testing

# CHAPTER 1

## INTRODUCTION

### 1.1 Background of Hate Speech Detection

Hate speech detection is an increasingly vital field in natural language processing (NLP), especially with the widespread use of social media and online platforms. Hate speech refers to any content that incites violence, discriminates against, or vilifies individuals or groups based on characteristics like race, gender, ethnicity, religion, or sexual orientation. As internet use continues to expand globally, the need for efficient systems to automatically detect and mitigate hate speech in real-time has become more urgent.

Existing systems largely rely on keyword-based filters and manual reporting, which often fail to account for the evolving nature of language, including slang, sarcasm, and regional dialects. Moreover, these traditional systems often lead to a high rate of false positives and negatives due to a lack of deep context understanding. This project aims to improve hate speech detection by leveraging state-of-the-art machine learning models, such as **BERT** (Bidirectional Encoder Representations from Transformers), alongside more traditional algorithms like **SVM**, **Naive Bayes**, and **Random Forest**.

### 1.2 Importance of Multilingual Detection

While many hate speech detection systems focus on English content, there is an increasing demand for multilingual capabilities. Online communities are diverse and global, with content being shared in various languages. The need for a system that can effectively classify hate speech across different languages becomes critical. By using advanced machine learning and NLP techniques, this project aims to build a **multilingual hate speech detection system** capable of analyzing content in multiple languages.

The integration of multilingual support ensures that hate speech is detected regardless of the language, helping to make online platforms safer for users around the world. This project will not only focus on textual content but will also extend its capabilities to include **audio** and **video** data, providing more comprehensive coverage of online content.

### 1.3 Project Objectives

The primary objectives of this project are:

- **To develop an advanced hate speech detection system** that goes beyond traditional keyword-based filters, using both modern and classical machine learning models.
- **To integrate multilingual capabilities** allowing the system to detect hate speech in various languages.
- **To support multiple data formats** such as text, audio, and video, for a more holistic approach to hate speech detection.
- **To evaluate and compare different models** like **BERT**, **Naive Bayes**, **SVM**, and **Random Forest**, ensuring that the best-performing models are utilized.
- **To create a user-friendly interface** using **Streamlit**, making the system easily accessible for real-time hate speech detection.

### 1.4 Problem Statement

Despite the growing awareness around online hate speech, current detection methods are often inadequate in addressing the complexities of modern

language, especially across different mediums (text, audio, video) and languages. The major challenges include:

- **Lack of Context Understanding:** Traditional keyword-based systems struggle to understand context, leading to inaccurate classifications.
- **False Positives and Negatives:** These systems often misclassify harmless content as hate speech or miss genuine instances of hate speech.
- **Multilingual and Multi-format Challenges:** Existing systems are usually limited to a single language or text-based formats, failing to address the diverse nature of online content.

This project aims to overcome these challenges by leveraging **BERT** for deep contextual understanding, combining it with other traditional machine learning models to create a **multilingual, multi-format hate speech detection system**. The system will also address issues like false positives and negatives, offering a more accurate, context-aware solution.

## 1.5 Scope of the Project

This project focuses on developing a system capable of detecting hate speech in **text**, **audio**, and **video** formats. The system will handle multiple languages and will be deployed as a **real-time application** through a **Streamlit interface**. Key features of the system include:

- **Text Classification:** Using models like **BERT** and **Random Forest** to classify content as hate speech or not.



- **Audio Processing:** Incorporating **Whisper** and **Speech Recognition** models to transcribe and analyze audio content.
- **Video Transcription:** Extracting audio from video files using **MoviePy** and transcribing them for hate speech analysis.
- **Multilingual Support:** The system will be able to detect hate speech across different languages, including non-English content.

## 1.6 Methodology

The methodology involves the integration of various machine learning models, each specialized for different aspects of the task. The process includes:

1. **Data Collection:** Data will be gathered from multiple sources, including social media posts, YouTube videos, and audio recordings. This data will be preprocessed to extract relevant features for model training.
2. **Model Training:** Several machine learning models will be trained using labeled datasets containing hate speech and non-hate speech examples. These models will include **BERT** for deep contextual understanding and **traditional models** like **SVM**, **Naive Bayes**, and **Random Forest** for comparison.
3. **Model Evaluation:** The models will be evaluated on metrics like **accuracy**, **F1 score**, **precision**, and **recall**. The system will also be tested for its **multilingual** capabilities, ensuring that it works effectively across different languages.

4. **User Interface Development:** A **Streamlit app** will be created to allow real-time interaction with the hate speech detection system. This interface will support text, audio, and video input.

## 1.7 Project Contributions

This project will contribute to the field of **online safety** by providing a scalable and adaptable solution to the growing problem of online hate speech. Key contributions include:

- A **multilingual hate speech detection system** capable of handling multiple data formats.
- Integration of **BERT** with traditional machine learning models to enhance classification accuracy.
- A user-friendly **real-time application** that can be used to monitor and report hate speech on online platforms.

## CHAPTER 2

### LITERATURE SURVEY

#### **TITLE 1:** *Multimodal Hate Speech Detection using Fine-Tuned Llama 2 Model*

**AUTHORS:** Sasidaran, K., & Geetha, J.

In this paper, the authors propose an innovative approach to hate speech detection by leveraging **multimodal inputs** (text and audio) and fine-tuning the **Llama 2 model** specifically for this task. They emphasize the challenges posed by **contextual nuances** in online content, where the meaning of certain words or phrases can vary depending on the tone or modality in which they are delivered. The authors highlight that existing hate speech detection systems often rely on **text-only** analysis, which fails to account for how hate speech can manifest in **audio** or **video** formats. By incorporating audio data, their system can detect **hate speech** in speech patterns, accents, and tonal changes that would be missed in text alone. The paper compares the **Llama 2** model with traditional approaches and shows that its fine-tuning for multimodal data significantly enhances accuracy, reducing both false positives and false negatives. The results indicate that **multimodal hate speech detection** offers a more robust solution than text-only models, especially in diverse online environments. The research also discusses the **real-world applicability** of this method, focusing on platforms where both text and speech content are prevalent. Additionally, the study examines the scalability of the model and its potential for **multilingual support**, ensuring that the approach can handle global content. This work pushes the boundaries of traditional **NLP models** by combining both **linguistic** and **acoustic features** for a more comprehensive hate speech detection system.

## **TITLE 2: *Recent advances in hate speech moderation: Multimodality and the role of large models***

**AUTHORS:** Hee, M. S., Sharma, S., Cao, R., Nandi, P., Chakraborty, T., & Lee, R. K. W.

This paper explores recent advancements in **hate speech moderation** by discussing the integration of **multimodal techniques** and **large models**. The authors investigate how traditional approaches, such as keyword filtering and manual flagging, fall short in moderating harmful online content due to their inability to capture **contextual variations** and **subtle linguistic cues**. The paper emphasizes that **large language models** (LLMs) like **BERT** and **GPT** have shown promising results in overcoming these limitations. However, the authors argue that these models perform even better when combined with **image**, **audio**, and **video** data in a **multimodal framework**. They describe how multimodal models enable a **deeper understanding** of content by analyzing both the **textual meaning** and the **context** in which it appears, such as tone in audio or visual cues in videos. The paper also outlines several key **challenges** with multimodal hate speech detection, including dealing with **data privacy** issues, managing the **biases** inherent in training data, and ensuring that systems are **scalable** for large volumes of user-generated content. The authors propose several strategies for improving hate speech detection using **multimodal large models**, such as **data augmentation**, **pre-training** on diverse datasets, and **fine-tuning** models for specific domains like **social media** or **online gaming platforms**. Ultimately, this research sets the stage for a future where **multimodal moderation systems** powered by large models can provide more accurate, context-aware, and ethical solutions for online content moderation.

### ***TITLE 3: Investigating the Predominance of Large Language Models in Low-Resource Bangla Language Over Transformer Models for Hate Speech Detection: A Comparative Analysis***

**AUTHORS:** Faria, F. T. J., Baniata, L. H., & Kang, S.

In this study, the authors analyze the performance of **large language models (LLMs)** in the context of **low-resource languages**, specifically **Bangla**, for **hate speech detection**. They highlight the challenges faced by many languages with limited annotated data, where traditional **transformer models** struggle to achieve the same performance as in high-resource languages like English. The authors demonstrate that **LLMs**, despite being more data-hungry, can still be fine-tuned to perform better in **low-resource scenarios** by employing various **transfer learning** techniques. The study compares several **transformer-based models** with **LLMs**, showing that fine-tuning the latter on smaller datasets allows them to outperform their predecessors in detecting hate speech in **Bangla**. By leveraging **cross-lingual knowledge transfer** and **data augmentation** strategies, the authors successfully overcome the data scarcity issue and improve detection accuracy. The research also discusses the potential for **multilingual detection** as a natural extension of LLMs, which can be applied to various **underrepresented languages** in hate speech detection. The paper concludes that **LLMs** offer a promising approach to enhancing hate speech detection, not just in **Bangla**, but also for other **low-resource languages**, thus expanding the reach and inclusivity of automated content moderation systems globally.

#### **TITLE 4: *Benchmarking public large language model***

**AUTHORS:** Malode, V. M.

Malode's doctoral dissertation investigates the **performance** and **application** of **public large language models (LLMs)** across a variety of tasks, including **hate speech detection**. The study provides a thorough **benchmarking** of several **publicly available LLMs**, comparing them with proprietary models to assess their **effectiveness** in tasks such as **sentiment analysis**, **toxic comment classification**, and **hate speech detection**. The dissertation evaluates the **strengths** and **limitations** of using **LLMs** for hate speech moderation, particularly when applied to real-time detection on **social media** platforms. The research highlights the **scalability** of LLMs for large datasets and discusses how **biases** in model predictions can result in both **false positives** and **false negatives**, thus impacting the accuracy and fairness of hate speech detection systems. Malode also delves into the ethical implications of using public LLMs, particularly in terms of **data privacy**, **model transparency**, and **accountability** in automated content moderation. The dissertation proposes several solutions for addressing these challenges, including better **data curation**, **bias mitigation**, and **adaptive retraining** of models. Additionally, the research sheds light on the **future potential** of **open-source LLMs**, arguing for more extensive collaborations between academia, industry, and government to improve hate speech detection at scale.

**TITLE 5: *A comprehensive cross-language framework for harmful content detection with the aid of sentiment analysis***

**AUTHORS:** Dehghani, M.

Dehghani's paper presents a **cross-language framework** for detecting **harmful content** in online platforms using **sentiment analysis**. The framework combines **multilingual sentiment analysis** with advanced machine learning techniques to identify **toxic comments**, **hate speech**, and other harmful online content across different languages. The author discusses how **language barriers** and **semantic differences** pose significant challenges to traditional **hate speech detection systems** that typically work well only in **English**. The framework utilizes **language-agnostic models** to improve detection across **low-resource languages**, allowing platforms to better detect harmful content from **non-English-speaking users**. The study also examines the role of **sentiment analysis** in determining the **emotional tone** behind a piece of content, which is crucial for identifying potentially harmful speech that may not be overtly offensive in text form but carries **negative sentiment**. The paper highlights the importance of **contextual understanding** and **cross-lingual transfer learning**, showing how these methods enhance the effectiveness of the system. It also discusses the application of this framework to real-time content moderation systems, including its use in **social media**, **forums**, and **news websites**. Finally, the paper explores future research directions for improving cross-lingual detection systems by expanding **multilingual datasets** and refining **sentiment analysis models**.

**TITLE 6: *MultiSentimentArcs: A novel method to measure coherence in multimodal sentiment analysis for long-form narratives in film***

**AUTHORS:** Chun, J.

**MultiSentimentArcs** is a novel method developed to analyze **coherence** in **multimodal sentiment analysis**, specifically for **long-form narratives in film**. While the primary application of this model is for sentiment analysis in movies, its principles can be extended to analyzing **hate speech** and **toxic content** across different media types, including **social media posts**, **videos**, and **audio**. The method measures how **emotional tone** and **sentiment** evolve over time in a **narrative**, using both **text** and **visual/audio cues**. This is particularly useful in analyzing content where **hate speech** is not overtly present but can still be **detected through tone** or **body language** in videos. The paper proposes that traditional sentiment analysis often misses **long-term sentiment arcs** and **narrative structure**, which are essential for accurately identifying the **context** in which harmful language or behavior occurs. By combining **textual sentiment analysis** with **visual cues** from films, the model helps detect **coherent emotional arcs** that would be difficult to identify through text alone. The research can significantly improve hate speech detection in multimedia content, where the harmful impact may not be limited to words but extended through other **multimodal cues**.



### **TITLE 7: *Brinjal: A Web-Plugin for Collaborative Hate Speech Detection***

**AUTHORS:** Hee, M. S., Singh, K., Si Min, C. N., Choo, K. T. W., & Lee, R. K. W.

The **Brinjal web plugin** introduces a **collaborative approach** to hate speech detection, allowing users to flag and report potentially harmful content in **real-time**. The plugin integrates seamlessly with **social media platforms** and **websites**, where users can participate in the detection and moderation of **hate speech**. The key innovation of Brinjal is its ability to allow users to **collaborate** in detecting hate speech while ensuring that the system can scale to handle **large volumes of user-generated content**. The paper outlines the architecture of the plugin, focusing on how **large language models (LLMs)** are used to process flagged content and **classify** it as hate speech or not. The authors also discuss the challenges of managing the **privacy concerns** that arise from involving users in the moderation process, as well as **balancing the accuracy** of the system to avoid false classifications. Additionally, the research highlights the ethical implications of community-driven moderation systems, focusing on the importance of ensuring **fairness** and **transparency** in automated hate speech detection.

### **TITLE 8: *A comprehensive review of multimodal large language models: Performance and challenges across different tasks***

**AUTHORS:** Wang, J., Jiang, H., Liu, Y., Ma, C., Zhang, X., Pan, Y., ... & Zhang, S.

This paper offers a **comprehensive review** of the **performance** and **challenges** associated with **multimodal large language models (MLLMs)** across various tasks, including **hate speech detection**. The authors provide an in-depth analysis of how multimodal models have improved the performance of automated content moderation systems by analyzing **text**, **images**, **audio**, and

**video** simultaneously. They discuss the advantages of multimodal approaches, such as enhanced **contextual understanding** and the ability to detect hate speech in non-verbal cues (e.g., facial expressions, body language). The review also highlights **performance challenges**, such as **data alignment** across modalities, the need for **huge computational resources**, and difficulties in training these models on diverse, **multilingual datasets**. The authors propose various solutions to improve the efficiency of MLLMs, such as **multi-task learning**, **transfer learning**, and **data augmentation**. The review concludes by discussing the future potential of MLLMs in **online content moderation**, especially for detecting **hate speech**, and how they could lead to more **accurate** and **scalable** detection systems.

**TITLE 9:** *Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects*

**AUTHORS:** Hadi, M. U., Al Tashi, Q., Shah, A., Qureshi, R., Muneer, A., Irfan, M., ... & Shah, M.

This survey comprehensively reviews **large language models (LLMs)**, examining their **applications**, **challenges**, **limitations**, and **future prospects**, with a particular focus on their use in **hate speech detection**. The authors explore the rise of LLMs in applications like **content moderation**, **customer support**, and **language translation**, highlighting their ability to understand and generate human-like text. The paper delves into the **ethical concerns** surrounding LLMs, such as the **biases** inherent in their training data, the **lack of transparency** in their decision-making, and the risks of **misuse**. The authors also discuss the limitations of current LLMs, including their dependency on massive datasets, **computational overhead**, and difficulty in understanding **context**. The paper concludes by proposing future directions for LLM research, emphasizing the need for **explainability**, **bias mitigation**, and **enhanced**

**ethical frameworks** for their deployment in **automated hate speech detection systems**.

**TITLE 10:** *Argumentative Stance Prediction: An Exploratory Study on Multimodality and Few-Shot Learning*

**AUTHORS:** Sharma, A., Gupta, A., & Bilalpur, M.

In this paper, the authors explore **argumentative stance prediction** using **multimodal approaches** and **few-shot learning**. The study demonstrates that **few-shot learning** can be used effectively to identify **argumentative stances** (e.g., **support**, **against**, **neutral**) in online discussions, particularly when training data is scarce. Although not specifically focused on hate speech, the findings are highly relevant, as the same techniques can be applied to detect **toxic or harmful content** in **arguments**. The paper discusses how **multimodal analysis** (incorporating **text**, **audio**, and **video**) can provide more nuanced insights into the **intent** behind a message. The authors argue that **few-shot learning** helps overcome data limitations by leveraging **pre-trained models** and adapting them to **new contexts** with minimal labeled data. This approach could be particularly useful for **hate speech detection**, where the **context** and **intent** of speech often play a significant role in determining whether content is harmful or not.

## CHAPTER 3

### SYSTEM ANALYSIS

#### 3.1 Existing System

The existing hate speech detection systems primarily rely on traditional **rule-based** approaches, **simple machine learning models**, and **manual content moderation**. These methods have been widely used over the years, but they are increasingly proving to be inefficient in dealing with the scale, complexity, and evolving nature of online content. One of the most common methods employed is **keyword-based filtering**, where the system scans content for predefined lists of **toxic or offensive words**. While this approach may work well in controlled environments with highly structured data, it fails to adapt to the dynamic and diverse nature of online communication. In addition, existing systems often depend on **manual moderators** who review flagged content, a process that is time-consuming, inconsistent, and often impractical in handling large volumes of content.

Traditional **machine learning models** like **Naive Bayes**, **Support Vector Machines (SVM)**, and **Logistic Regression** are also commonly employed in hate speech detection. These models are typically trained on labeled datasets, where hate speech and non-hate speech examples are used to teach the algorithm to differentiate between the two. However, these models often struggle to capture the **context** and **subtleties** of language. They rely heavily on **feature extraction** methods that focus on individual words, phrases, or syntactic structures without taking into account the broader **semantic meaning** of the text or the **intent** behind the words. This limitation often leads to

**misclassifications**, especially in cases where the language is **sarcastic**, **ironic**, or uses **coded language** to convey hate speech indirectly.

Existing systems also fail to **understand multimodal content**, where hate speech may not only appear in text but also in **audio** or **video**. For instance, a **spoken sentence** may convey hate speech through the tone of voice or an **offensive gesture** in a video. Current systems are not designed to detect these subtleties, which limits their effectiveness in detecting hate speech across different media types.

### 3.1.1 Disadvantages

The **existing systems** for hate speech detection have several notable disadvantages that make them less effective in modern, dynamic online environments.

1. **Lack of Contextual Understanding:** A primary limitation of current systems, especially **keyword-based filtering**, is their inability to understand the **context** in which certain words are used. Hate speech is often highly contextual, and the same word or phrase may have different meanings depending on the situation. For example, the phrase “I hate this” might be used to express frustration or dislike towards an object, but it can also carry hateful connotations in the context of human beings or marginalized groups. Current systems fail to distinguish between these two uses, leading to **false positives** or **false negatives** in classification. Furthermore, many traditional models also struggle to detect **sarcasm**, **irony**, and **implicit hate speech**, which are often more difficult to identify without a deeper understanding of the conversation’s context.
2. **False Positives and False Negatives:** One of the significant challenges with existing hate speech detection systems is their **high error rate** in

both **false positives** and **false negatives**. False positives occur when content is incorrectly flagged as hate speech, even though it might be benign, leading to **unwarranted censorship** or **over-moderation**. On the other hand, false negatives occur when harmful content is overlooked, allowing **hate speech** to go undetected and potentially spread. This happens because traditional systems lack the **semantic understanding** needed to differentiate between harmful and non-harmful content effectively. The result is a **high volume of incorrect classifications**, which diminishes the overall reliability and accuracy of the system.

3. **Inflexibility and Dependence on Keyword Lists:** The keyword-based systems in use today are highly **inflexible**. They rely on static, predefined keyword lists to detect hate speech, but **language evolves** constantly. New **slang**, **neologisms**, and **coded language** used by hate groups often bypass these systems. Without continuous updates and manual intervention to refresh keyword lists, these systems become obsolete. Moreover, this dependence on keyword lists does not account for **contextual meaning**—where the same word can have completely different implications depending on the surrounding text or situation.
4. **Scalability and Operational Challenges:** Given the sheer volume of content generated on **social media platforms** and **online forums**, current **manual moderation** systems cannot scale effectively. Relying on human moderators to review flagged content results in significant delays and inconsistencies. This process is also highly **resource-intensive**, requiring substantial investments in terms of time, personnel, and operational costs. For large platforms with millions of posts being generated every second, **manual moderation** is simply not viable. Additionally, the **latency** involved in this process leads to a poor user experience and can make it

difficult to respond to **real-time hate speech**.

5. **Multimodal and Multilingual Limitations:** Another key limitation is that existing systems are primarily focused on **text-based** content and are not designed to handle **multimodal data** such as **audio** or **video**. For example, hate speech can manifest through tone, inflection, or **visual cues**, which are missed by text-only models. Furthermore, these systems are typically limited to a specific language or set of languages, meaning they are ineffective for **multilingual** content, which is increasingly common on global platforms. As the internet becomes more diverse and interconnected, the inability to handle **multilingual content** further exacerbates the problem.

### 3.2 Proposed System

The proposed system aims to address the limitations of the existing hate speech detection methods by incorporating **state-of-the-art models**, including **multimodal and multilingual capabilities**, to enhance the accuracy and adaptability of hate speech detection systems.

1. **Multimodal Integration:** One of the key innovations of the proposed system is the integration of multiple modalities—**text**, **audio**, and **video**—into a single framework for hate speech detection. By analyzing both the **text** of the message and **non-verbal cues** such as **tone** or **facial expressions**, the system can provide a more holistic and accurate detection of hate speech. For example, a **spoken sentence** may carry harmful intent when heard with a particular tone, or a **video** may convey hate speech through offensive gestures or body language. The system utilizes **advanced multimodal models** to combine these different types

of data into a unified framework, improving both the **accuracy** and **contextual understanding** of the detection process.

2. **Advanced Language Models:** The proposed system will leverage **state-of-the-art large language models** (LLMs) such as **BERT** and **GPT** for text analysis. These models are capable of understanding the deep context of the content, recognizing not just the presence of harmful words but also the **intent** behind them. Furthermore, the system will incorporate **transfer learning** techniques to **fine-tune** these models on **domain-specific datasets**, improving their ability to detect hate speech in specific communities or contexts. This allows the model to adapt to the evolving nature of language and emerging slang, reducing the **need for constant updates** and improving **adaptability**.
3. **Multilingual and Cross-Cultural Capabilities:** The system will include **multilingual support**, allowing it to handle content in various languages. This is essential for global platforms, where users communicate in different languages. By incorporating **cross-lingual models** and training the system on **diverse datasets**, it can identify hate speech in **non-English languages**, expanding its reach and effectiveness. The system will also be capable of understanding **cultural nuances**, which are important when analyzing content from users in different regions. This feature makes the proposed system highly scalable and effective in a global context.
4. **Real-Time Detection and Scalability:** The system will be designed for **real-time detection** of hate speech, enabling platforms to automatically flag harmful content as it is posted. With the incorporation of **deep learning techniques**, the system will be able to process and analyze large



volumes of data quickly and accurately, making it suitable for use on large-scale platforms like **social media** and **forums**. The system will also be highly **scalable**, capable of handling the ever-increasing amount of content generated across the internet without significant delays or loss of accuracy.

### 3.2.1 Advantages

The proposed system offers several advantages over existing hate speech detection systems:

1. **Enhanced Accuracy and Reduced Errors:** By combining text, audio, and video data, the proposed system can better understand the **context** and **intent** of the content, reducing **false positives** and **false negatives**. This comprehensive approach ensures that harmful content is accurately detected while minimizing unnecessary censorship of harmless content.
2. **Scalability:** The system is designed to be scalable, capable of handling large volumes of content in real time. With the use of **distributed computing** and **cloud-based architecture**, the system can be deployed across platforms of all sizes, from small communities to global social networks.
3. **Adaptability to Evolving Language:** The use of **large language models** and **transfer learning** ensures that the system remains adaptable to the evolving nature of language. The system can continuously improve by being retrained on new data, ensuring that it remains effective in detecting **emerging hate speech patterns** and slang.

4. **Global Reach:** With its **multilingual capabilities**, the system is well-suited for **global platforms** where users communicate in various languages. It ensures that **hate speech detection** is consistent and reliable, regardless of the language or cultural context in which the content is generated.

### 3.3 Architecture Diagram

The architecture of the proposed system consists of several key components:

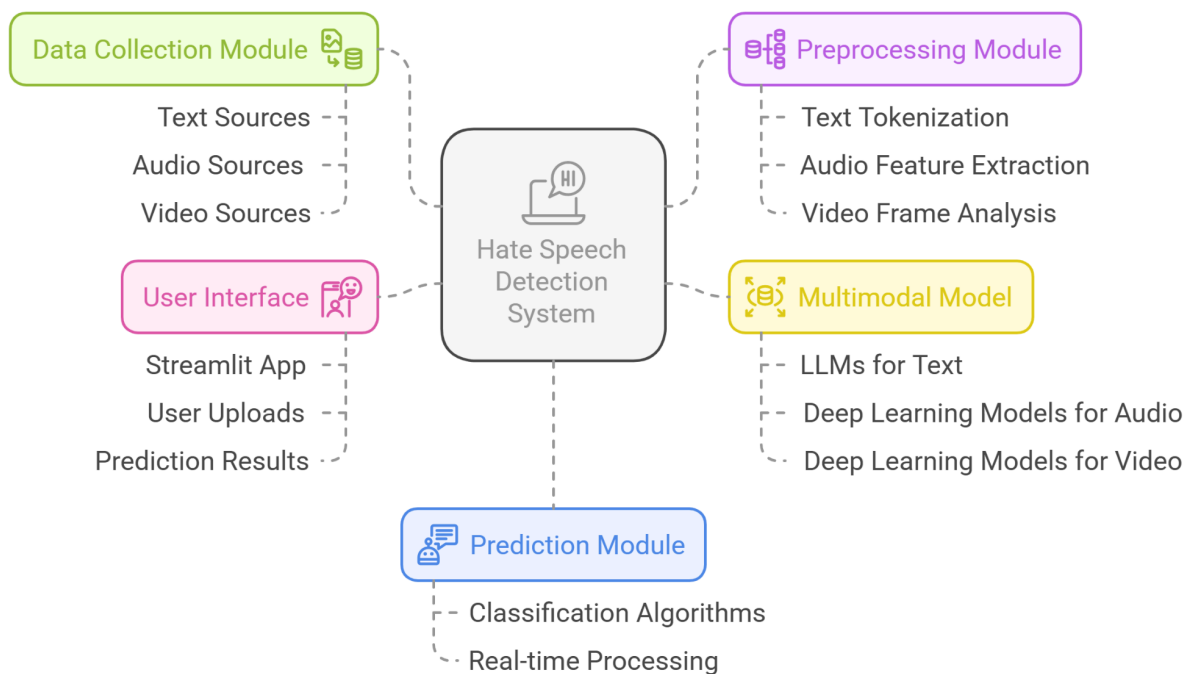


Figure 3.1: System Architecture

- **Data Collection Module:** Gathers content from various sources (text, audio, video) across social media platforms.
- **Preprocessing Module:** Preprocesses the data, including text tokenization, audio feature extraction, and video frame analysis.

- **Multimodal Model:** Analyzes the combined data (text, audio, video) using a **multimodal neural network** that incorporates LLMs for text and deep learning models for audio and video data.
- **Prediction Module:** Classifies the processed data as **hate speech** or **non-hate speech**.
- **User Interface:** Provides a **Streamlit app** for users to interact with the system, upload content, and receive real-time predictions.

The architecture ensures that the system is capable of handling multiple data formats and providing accurate, context-aware hate speech detection at scale.

## CHAPTER 4

### SYSTEM REQUIREMENTS

#### 4.1 Hardware Requirements

The hardware requirements for the hate speech detection system are critical for ensuring **efficient** and **fast processing**, especially given that the system will handle large datasets, perform **real-time analysis**, and use **machine learning models** that can be computationally intensive. The following hardware components are necessary:

1. **Processor (CPU):**

A **multi-core processor** with at least **8 cores** is essential for parallel processing, enabling the system to handle large-scale computations and multi-threaded tasks efficiently. For optimal performance, an **Intel i7** or **AMD Ryzen 7** processor, or higher, should be used. The high number of cores is especially important when running machine learning tasks and processing **multimodal data** (text, audio, video) in parallel.

2. **Graphics Processing Unit (GPU):**

Since the system utilizes **deep learning models** such as **BERT** and other transformer-based architectures, a **GPU** with **CUDA support** is crucial for accelerating model training and inference. A **NVIDIA GTX 1080** or better (e.g., **RTX 3060**, **RTX 3080**) would significantly speed up tasks like model inference, video frame processing, and audio analysis, which require substantial computational resources.

3. **Memory (RAM):**

For **multimodal processing** and handling large datasets in **real-time**, a

minimum of **16 GB of RAM** is required, although **32 GB** or more is recommended for smooth performance, particularly if the system needs to run multiple models simultaneously. Sufficient RAM ensures that the system can process large volumes of data without slowing down.

#### 4. **Storage:**

The system will require both **SSD** (Solid State Drive) and **HDD** (Hard Disk Drive) storage. The **SSD** should be used for the **operating system** and **application files**, enabling fast data access and reducing load times. The minimum required SSD size is **500 GB**. For long-term storage of **training datasets** and **model outputs**, an **HDD** with **2 TB** or more capacity is necessary, especially for large video and audio files.

#### 5. **Network Connectivity:**

Since the system may need to handle **real-time data feeds** and provide online **feedback** via **Streamlit** or other web interfaces, a **high-speed internet connection** is recommended. A **minimum of 10 Mbps** download and **upload speeds** are recommended for seamless real-time interactions, especially when processing media-heavy content like video or audio files.

#### 6. **Additional Peripherals:**

- **Webcam and Microphone** for testing and verifying the system's ability to process live video and audio inputs.
- **Display:** At least a **Full HD monitor** (1920x1080 resolution) to ensure clarity during system interaction, particularly when displaying visual feedback or results.

These hardware requirements ensure that the system will run optimally, especially for **large-scale social media platforms** or when processing large multimedia datasets.

## 4.2 Software Requirements

To successfully implement the proposed hate speech detection system, a variety of **software** tools, **libraries**, and **platforms** are necessary. These components will enable smooth development, deployment, and maintenance of the system.

### 1. Operating System:

The system will run on **Windows**, **Linux**, or **macOS**. However, Linux-based systems are recommended for development and deployment because of their flexibility, better support for machine learning libraries, and greater compatibility with **NVIDIA GPUs**. Additionally, Linux offers a better environment for handling large-scale data and running deep learning models with **CUDA**.

### 2. Programming Languages:

The primary programming language used will be **Python** due to its extensive support for **machine learning**, **data analysis**, and **natural language processing**. Python libraries such as **PyTorch**, **TensorFlow**, and **Transformers** will be utilized for training and inference with large language models like **BERT**. For the **frontend**, **JavaScript** and **HTML/CSS** will be used, especially with **Streamlit** to develop an interactive user interface for real-time hate speech detection.

### 3. Machine Learning Libraries and Frameworks:

The system will leverage several powerful machine learning libraries:

- **PyTorch**: For building and training deep learning models, including BERT and multimodal networks.
- **TensorFlow/Keras**: To support model training and deployment in a more structured framework.
- **Transformers (Hugging Face)**: For loading, fine-tuning, and running transformer-based models like **BERT** and **Llama-2**.
- **Scikit-learn**: For implementing traditional machine learning models like **SVM**, **Random Forest**, and **Logistic Regression**.
- **NLTK** and **spaCy**: For **natural language processing** tasks such as tokenization, stopword removal, and dependency parsing.
- **OpenCV**: For processing and analyzing video frames to detect hate speech through visual cues.

#### 4. Multimodal Libraries:

- **MoviePy**: To extract audio from video files for transcription and subsequent hate speech detection.
- **SpeechRecognition**: For converting audio into text.
- **Whisper** (by OpenAI): A model for **speech-to-text** transcription, particularly useful for understanding speech in multiple languages.
- **Librosa**: To process audio features such as pitch, tempo, and tone, which can be used to detect the sentiment or emotional context in spoken content.

#### 5. Web Development Frameworks:

- **Streamlit**: For building the **user interface** (UI) and deploying the model in a web application, allowing users to upload **text**, **audio**, or **video** files and receive real-time predictions.

- **Flask/Django**: In case additional backend services or APIs are needed, Flask or Django can be used to provide a **REST API** for handling requests.

## 6. Database and Data Storage:

- **SQLite** or **PostgreSQL**: For storing **user data** (e.g., account details, session history) and any other metadata required by the system. If the system needs to store user-uploaded files, **Amazon S3** or **Google Cloud Storage** can be used for scalable, cloud-based storage solutions.

## 7. Cloud Services and Deployment:

- **AWS/GCP/Azure**: Cloud platforms will be used for **model hosting, data storage, and scalable deployment**. Tools like **AWS Lambda** or **Google Kubernetes Engine (GKE)** can be employed to ensure that the system scales effectively as user demand increases.
- **Docker**: To containerize the application, ensuring that it runs consistently across different environments, whether during development or in production.

### 4.3 Software Specification

The **software specification** outlines the specific software features, functionalities, and interactions between different components of the system. This section provides a detailed overview of the software architecture and the individual components involved in the system's development.



### 4.3.1 Front-End

The front-end of the hate speech detection system will serve as the **user interface (UI)** for interacting with the system. It will allow users to upload content, interact with the hate speech detection models, and view the results of their interactions. The front-end will be built using **Streamlit**, which provides a simple yet powerful platform for rapid prototyping and building interactive web applications.

Key components of the **front-end**:

#### 1. **User Input Section:**

- **Text Input:** Users will be able to input text directly into the system, which will then be processed by the language models for hate speech detection.
- **Audio and Video Upload:** Users can upload audio or video files, which will be processed to extract speech and detect potential hate speech through transcription and contextual analysis.
- **Live Audio Recording:** The system will allow users to record live audio through the microphone, transcribe it, and analyze the content in real time.

#### 2. **Processing Indicators:**

- A **loading spinner** or progress bar will be displayed during the processing of audio/video files to inform the user of the system's

status.

### 3. Result Display:

- Once the content has been processed, the system will show the results of the **hate speech classification**, including labels such as **Hate Speech** or **Not Hate Speech**, along with a confidence score.
- **Color-coded feedback** will be used, such as green for "Not Hate Speech" and red for "Hate Speech," ensuring that users can easily interpret the results.

### 4. User Authentication:

- The system will include a **login/signup page** for users, allowing them to create an account, log in securely, and access their history of past submissions.

### 5. Real-Time Feedback:

- The **Streamlit interface** will provide **real-time feedback** to the users, especially when they interact with the system via live recording, enabling a **seamless user experience**.

The front-end will ensure that users can easily upload content, receive predictions, and interact with the system without requiring deep technical knowledge.

## CHAPTER 5

### MODULES DESCRIPTION

#### 5.1 User Authentication Module

The **User Authentication Module** plays a crucial role in ensuring the security and privacy of users within the system. This module allows users to create accounts, log in, and access the **hate speech detection system** securely. The authentication process will involve two primary functionalities: **signing up** and **logging in**.

During the **sign-up process**, users will input essential information such as their **name**, **email**, **age**, **sex**, and a **password**. The password will be encrypted for security purposes before being stored in the database. Once the account is created, users can log in using their **email** and **password**. If the login is successful, users will be granted access to the system and its features. If there's an issue with the credentials, an error message will prompt the user to try again.

Additionally, this module supports **session management** to track logged-in users and ensure that their data and preferences persist during their interaction with the system. Using **JWT (JSON Web Tokens)** or **sessions** in Python's **Flask/Django framework**, the system securely stores user sessions to avoid unauthorized access.

The **User Authentication Module** will ensure that users can safely and easily interact with the hate speech detection system while maintaining confidentiality and security.

## 5.2 Hate Speech Detection Module

The **Hate Speech Detection Module** is the core component of the system and is responsible for analyzing the content provided by users and classifying it as either **hate speech** or **non-hate speech**. This module will utilize several advanced **machine learning models**, including **BERT (Bidirectional Encoder Representations from Transformers)** for text analysis and other deep learning models for analyzing **audio** and **video** content.

The detection process begins with data preprocessing, where the input data (text, audio, or video) is cleaned and transformed into a suitable format for analysis. For **text**, this might involve tokenization, stopword removal, and other NLP techniques. For **audio**, speech-to-text transcription is done using models like **Whisper** or **Google Speech Recognition**, and for **video**, audio tracks are extracted and then processed.

Once the data is ready, the **machine learning model** performs the classification task. The system will apply **BERT**, trained on a large-scale dataset containing both **hate speech** and **non-hate speech** examples, to evaluate the content. The model checks for context, intent, and emotion to accurately classify the text. For **audio** and **video**, additional models may be used to detect tone, inflection, or non-verbal cues such as body language or gestures, enhancing the system's ability to detect hate speech in multimodal data.

Finally, the module outputs the classification results, which are then presented to the user, typically with a confidence score and a visual cue (e.g., red for "Hate Speech," green for "Not Hate Speech").

### 5.3 Multimodal Integration Module

The **Multimodal Integration Module** is designed to integrate and process data from multiple sources, such as **text**, **audio**, and **video**. The goal of this module is to provide a more comprehensive and accurate analysis by combining the insights gained from each type of input. Hate speech detection can often be more complex when it involves spoken words or video, where tone of voice, facial expressions, and body language also play a significant role in determining intent.

This module is responsible for receiving **input data** from the user, whether it is in **text**, **audio**, or **video** format, and preparing it for analysis by the appropriate models. If the input is **text**, it is passed to the **text processing model** (e.g., **BERT**). For **audio**, it is converted to text using speech recognition tools, and for **video**, the audio is extracted, and both the **visual and audio data** are processed for a multimodal analysis.

The **multimodal integration** allows the system to detect **implicit hate speech**, which may not always be explicitly expressed in words but conveyed through **tone**, **gesture**, or **body language**. This makes the system more robust and adaptable to various forms of online communication, such as live streams, social media posts, or video blogs.

The **Multimodal Integration Module** ensures that the system can handle various data formats, making it more effective in detecting hate speech across different types of content.

## 5.4 Real-Time Prediction Module

The **Real-Time Prediction Module** is designed to analyze content and provide hate speech classification on the fly. This module is crucial for applications where immediate feedback is necessary, such as during **live audio or video recordings** or while users are interacting with the platform. The key goal of this module is to ensure that hate speech is detected instantly, with minimal delay.

The **real-time functionality** of the module relies on optimized models and an efficient backend system. When users upload content, the module processes the data in **real-time** by utilizing **pre-trained models** (e.g., **BERT**, **SVM**, **Random Forest**) to classify the content. For text inputs, the module performs immediate classification based on the latest model in place. For audio or video, **audio transcription** happens quickly via speech-to-text tools, followed by hate speech detection.

The module needs to handle high volumes of incoming data, so **cloud-based infrastructure** like **AWS Lambda** or **Google Cloud Functions** will be utilized to ensure scalability and reliability. The system will notify users with a **confidence score** for the classification result, along with a **real-time display** that updates automatically without the need for the user to refresh or wait for the results.

This module is particularly useful for applications where users engage in **live streaming**, **real-time social media communication**, or **moderation tasks**, enabling **instant detection** of harmful content.

## 5.5 Feedback and Reporting Module

The **Feedback and Reporting Module** allows users to interact with the system by providing feedback on the results of the **hate speech detection**. This feedback can be used to improve the accuracy of the system and provide users with a more personalized experience.

When a user receives the hate speech detection result (e.g., "Hate Speech" or "Not Hate Speech"), they have the option to **dispute** the result if they believe it is incorrect. The module captures this feedback and stores it in the system. The user can provide a **reason** for their feedback, which helps the system understand the context better and adapt over time.

This module also includes a **reporting feature** where users can flag **misclassified** content or **appeal** a decision if they believe their content was wrongly flagged as hate speech. The system uses this feedback to fine-tune the models and **retrain** them periodically. This helps improve the **accuracy** and **reliability** of the system and ensures that it evolves with the changing nature of language and online discourse.

Moreover, the module includes a **moderation interface** where administrators can review flagged content, view the feedback provided by users, and adjust the system settings based on user input.

## 5.6 Admin Control and Management Module

The **Admin Control and Management Module** is designed to provide platform administrators with the ability to manage users, monitor system performance, and adjust system parameters to ensure smooth operation. This module allows administrators to oversee the functioning of the **Hate Speech Detection System** and intervene when necessary.

Key features of this module include:

1. **User Management:** Administrators can view, modify, or deactivate user accounts as needed. They can also manage **user permissions**, ensuring that only authorized personnel have access to sensitive data or administrative functions.
2. **System Performance Monitoring:** Admins can monitor the **performance** of the detection system, such as the speed of processing, accuracy of predictions, and volume of flagged content. They can access **logs** and detailed reports to identify potential issues.
3. **Model Management:** Admins can update and manage the **machine learning models** used in the system. This includes **retraining models** with new data, evaluating **model performance**, and applying **patches** to address model drift or biases.
4. **Content Moderation:** Admins can review and resolve flagged content, especially in cases where users report errors or submit disputes. This ensures that the system remains fair and accurate while maintaining a **non-biased** approach to content moderation.



By providing full administrative control over system operations, the **Admin Control and Management Module** ensures that the system remains **reliable**, **scalable**, and **adaptable** to future requirements.

## CHAPTER 6

### SYSTEM DESIGN

#### 6.1 UML Diagrams

Unified Modeling Language (UML) diagrams are essential tools for visualizing the design of a system. They provide clear representations of the system's structure, functionality, and flow of processes. In the context of this hate speech detection system, UML diagrams help in breaking down complex interactions and components, making it easier to understand the system's architecture and how different modules interact with each other. The key UML diagrams that will be used for this system include the **Use Case Diagram**, **Sequence Diagram**, **Activity Diagram**, and **Class Diagram**. Each of these diagrams serves a specific purpose in depicting different perspectives of the system.

##### 6.1.1 Use Case Diagram

The **Use Case Diagram** provides an overview of how the system will interact with external users (actors) and other systems. It is a high-level diagram that helps in identifying the primary functionalities or **use cases** that the system will perform. For the hate speech detection system, the **actors** could include **end users**, **administrators**, and potentially external systems like social media platforms or third-party APIs for **speech-to-text** services.

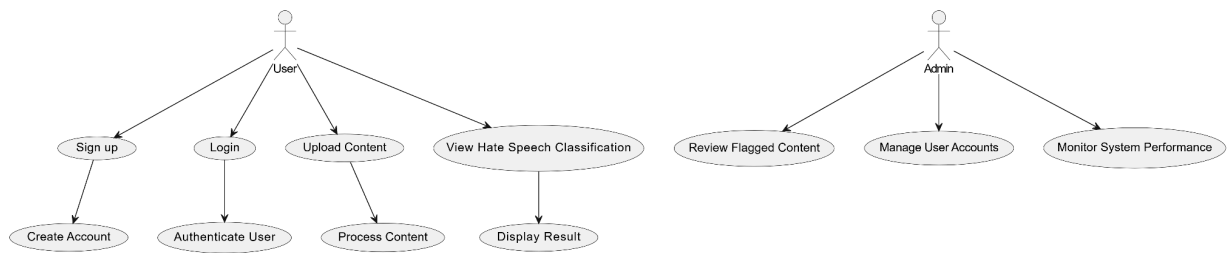


Figure 6.1:Use Case Diagram

In this diagram, each **use case** represents a specific action or service that the system offers. For example, **signing up**, **logging in**, **uploading content** (text, audio, video), and **receiving hate speech classification** results are some of the use cases for the user. The **administrator** may have additional use cases like **reviewing flagged content**, **managing user accounts**, or **monitoring system performance**. These actors will interact with the system in various ways depending on their roles. The Use Case Diagram provides a **clear blueprint** of the system's **functional scope**, ensuring all required functionalities are captured and accounted for during development.

The **Use Case Diagram** is particularly useful during the initial stages of system development as it helps to align the requirements with the actual system design. It also facilitates communication between stakeholders (e.g., developers, project managers, clients) by offering a simplified visualization of the user-system interactions.

## 6.1.2 Sequence Diagram

The **Sequence Diagram** is a detailed representation of how objects or components in the system interact with each other over time. It shows the sequence of events or messages passed between objects to complete a particular task or process. In the context of the hate speech detection system, a Sequence Diagram could be used to depict the flow of a **user's interaction** with the system, from the moment they upload content to receiving a hate speech classification result.

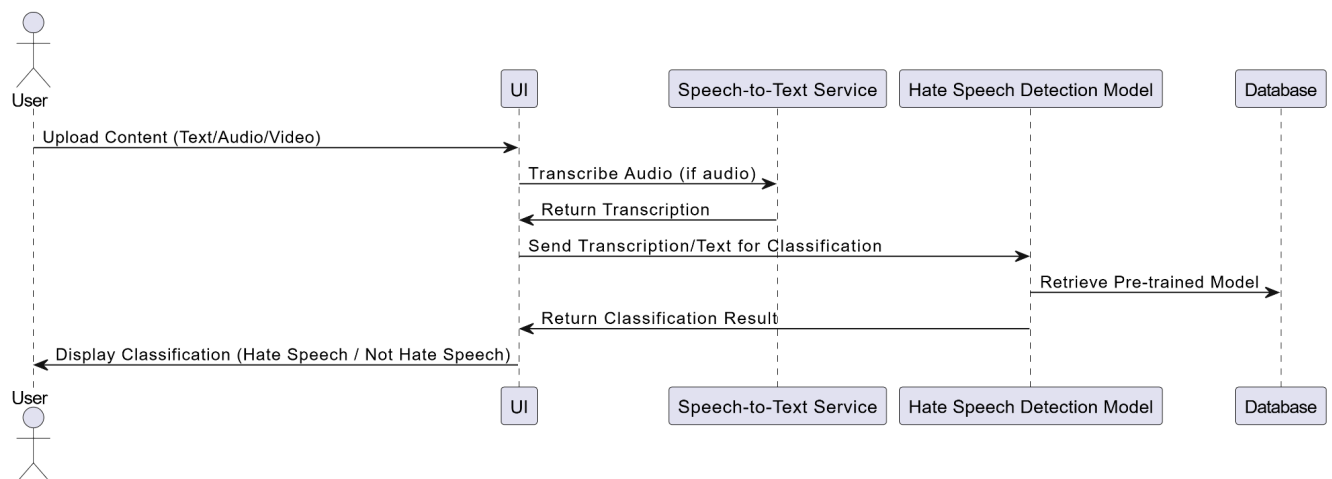


Figure 6.2: Sequence Diagram

For example, when a user uploads an audio file, the **Sequence Diagram** would illustrate how the system processes the file: First, the system sends a request to **speech-to-text** services for transcription, then sends the resulting text to the **hate speech detection model** for classification. The sequence of method calls, responses, and interactions between the **user interface**, **backend services**, and **machine learning models** will be clearly shown in this diagram.

The **Sequence Diagram** is important because it highlights the **timing** and **order** of interactions, ensuring that each component communicates efficiently. It also helps identify any **potential bottlenecks** or points where delays might occur,

such as in real-time transcription or classification, and offers a clear path for debugging and optimization.

### 6.1.3 Activity Diagram

The **Activity Diagram** focuses on illustrating the workflow or **control flow** of activities within the system. It highlights how different **actions** and **decisions** occur sequentially or concurrently to complete a given process. For the hate speech detection system, an **Activity Diagram** might show the flow of actions when a user uploads content, starting from the initial input phase through to the final classification result.

For example, when a user uploads a text file, the diagram will depict the series of actions: first, the file is uploaded to the system; then, the content is processed (cleaned, tokenized); next, the **hate speech detection model** is called for analysis, followed by decision-making steps based on the model's prediction (whether it is **hate speech** or **non-hate speech**). The diagram also helps illustrate decisions, such as **whether the user needs to provide feedback** on the system's classification result, or if **administrator intervention** is required when certain thresholds are met.

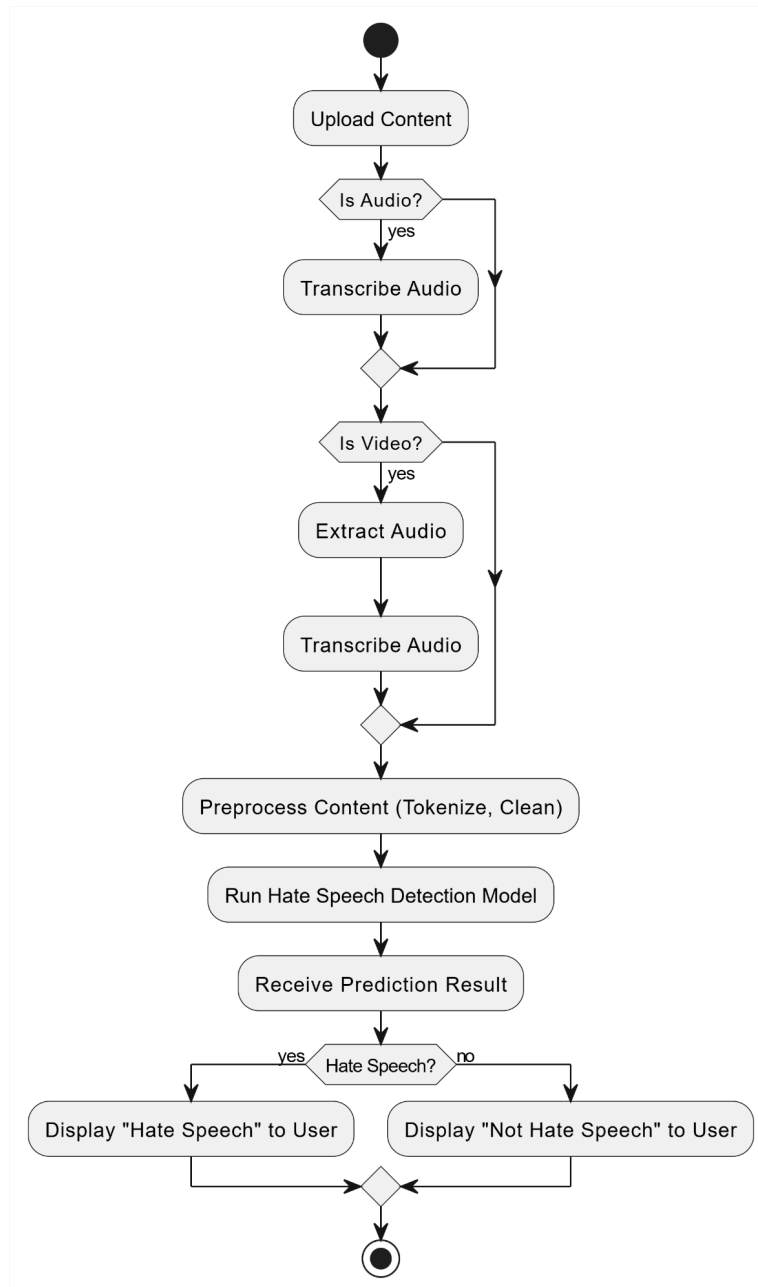


Figure 6.3: Activity Diagram

The **Activity Diagram** is a key tool for visualizing the **overall flow** of the system's processes and ensuring that all possible pathways (including decision points) are accounted for. This diagram is particularly useful for understanding the **logical progression** of user actions and system responses and can highlight areas that might need further optimization or refinement.

### 6.1.4 Class Diagram

The **Class Diagram** is a structural diagram that represents the **objects** in the system and their **relationships**. It provides a detailed view of the **system's static structure**, focusing on the classes, attributes, methods, and the associations between them. In the context of the hate speech detection system, the Class Diagram could depict key components such as **User**, **Content**, **HateSpeechModel**, and **Administrator**.

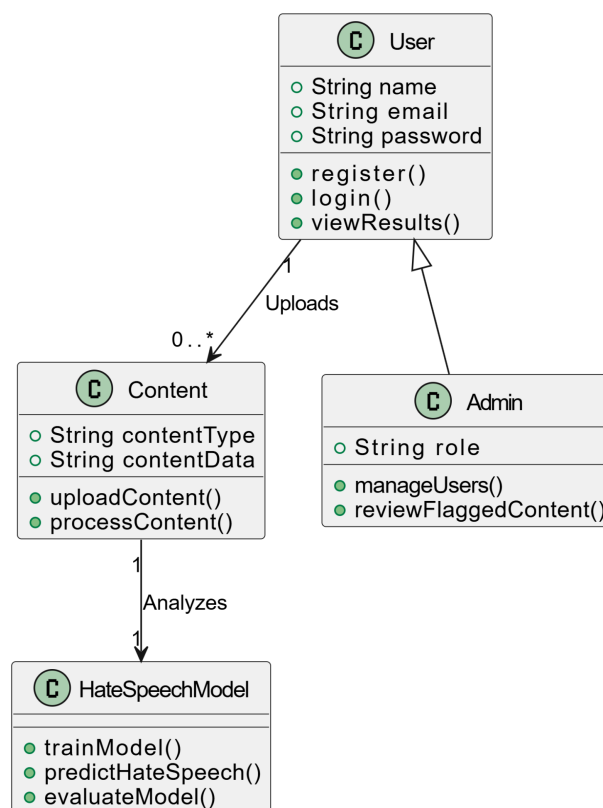


Figure 6.4: Class Diagram

For example, the **User** class would include attributes like **name**, **email**, and **password**, along with methods for **registering**, **logging in**, and **viewing results**. The **HateSpeechModel** class would include methods for **training**, **predicting**, and **evaluating** hate speech, along with associated attributes like **model parameters** and **training data**. Relationships between classes could include

inheritance (e.g., the **Administrator** class inheriting from the **User** class) or associations (e.g., a **User** may have multiple **uploadedContent** entries).

The **Class Diagram** serves as a blueprint for the **backend** structure of the system. It is vital for defining how different objects in the system interact and ensuring that **object-oriented design principles** are followed. The diagram is especially useful for developers and designers, as it helps them understand the **relationships** between system components, the **flow of data**, and how different objects will collaborate to achieve the system's functionality.

This diagram is also critical when implementing system changes or additions. By maintaining a clear understanding of class relationships and object structures, developers can ensure that new features or modifications are compatible with the existing system architecture.



## CHAPTER 7

### TESTING

Testing is an essential part of the software development lifecycle, ensuring that the system functions as expected and meets all necessary requirements. It helps to identify **bugs** and **issues** early in the development process, improving the overall quality and stability of the system. For the hate speech detection system, different types of testing will be employed at various stages of development to ensure its functionality, performance, and security. Below are the different types of testing that will be implemented:

#### 7.1 Types of Testing

The following types of testing will be used to ensure that the hate speech detection system works reliably and meets the specifications:

##### 7.1.1 Unit Testing

Unit testing is one of the most fundamental types of testing, where individual components or units of the system are tested in isolation to verify their correctness. The goal of **unit testing** is to ensure that each **function** or **method** within the application works as intended. For instance, in the hate speech detection system, **unit tests** will be written to test individual functions within the modules like **content preprocessing**, **speech-to-text conversion**, **model inference**, and **result classification**.

Each function or method will be tested independently with a set of **known inputs** and **expected outputs**. For example, the **text tokenization** function will be tested with sample text inputs to ensure that it splits the text into correct tokens and handles edge cases like **empty strings** or **special characters**. Similarly, functions like **audio transcription** and **hate speech classification**

will be tested to ensure that they handle different types of input correctly, such as **valid audio files**, **non-English speech**, and **noisy audio**.

Unit testing is vital because it allows developers to catch bugs early in the development process, making it easier to fix issues before they propagate to other parts of the system. Testing frameworks like **PyTest** or **unittest** in Python will be used to automate the execution of unit tests, ensuring that all functions work correctly with every code change.

Moreover, unit tests provide **regression testing** benefits by helping developers ensure that new code changes do not break existing functionality. With a well-structured set of unit tests, the hate speech detection system can evolve without fear of inadvertently affecting previous functionality, providing higher confidence in the software's reliability.

In the context of the hate speech detection system, unit tests will also help verify the correct integration of **machine learning models**. While testing the models themselves can be complex, testing individual prediction functions, ensuring they produce correct classifications for test data, is essential. This ensures that the model is consistently generating results as expected.

Unit testing also makes the code more **maintainable** and **scalable**, as developers can quickly identify and address defects in specific parts of the system without affecting the overall project. Each unit test serves as documentation of the expected behavior of the function or method, helping new developers understand the system's structure more easily.

### 7.1.2 Integration Testing

**Integration testing** is conducted after unit testing to ensure that different modules or components of the system work together as expected. While unit testing focuses on isolated pieces of functionality, integration testing focuses on how these pieces interact and function when combined. The hate speech detection system consists of multiple interconnected modules such as **audio transcription**, **text preprocessing**, **model inference**, and **feedback systems**, all of which must work together seamlessly.

In integration testing, different modules are combined, and data is passed between them to ensure smooth integration. For example, once the **audio transcription module** has transcribed the speech, the **text preprocessing module** should properly clean and tokenize the text. Then, the **hate speech detection model** should classify the processed text correctly, and finally, the **results** should be displayed to the user through the front end.

A critical aspect of integration testing in this system will be checking for **data flow issues** between modules. For instance, if the text data isn't correctly passed from the **speech-to-text service** to the **preprocessing module**, the system could fail to recognize hate speech. Similarly, if the **model's prediction** isn't properly passed to the **user interface**, the results won't be visible to the user. These data transfer points must be tested to ensure that the system functions smoothly and reliably.

Additionally, **error handling** is a key component of integration testing. The system should gracefully handle situations where one module fails, such as when the **speech-to-text service** cannot transcribe audio properly or when the **model** fails to provide a valid classification. Integration tests will simulate such failures and ensure the system behaves as expected, by showing appropriate error messages or fallback options for the user.

Another aspect of integration testing involves validating the integration of external services, such as **cloud storage** or **third-party APIs** used for speech recognition or model hosting. These services must be integrated correctly into the system, with appropriate **API calls** and **data handling mechanisms**. Testing the interaction between the internal system and external services ensures that the system can interact with third-party components as expected.

Integration testing will also help uncover issues related to **scalability**. When the system is handling large volumes of data, such as many users uploading audio or video files for processing, integration testing will ensure that all components can handle such loads without failures.

### 7.1.3 System Testing

System testing is the final level of testing where the entire system is tested as a whole to ensure that it meets the specified requirements and functions correctly. Unlike unit testing and integration testing, which focus on individual components and their interactions, **system testing** evaluates the entire application from end to end.

In the context of the hate speech detection system, system testing will involve testing the entire workflow, from the user **signing up** and **logging in**, to **uploading content** (text, audio, or video), **processing the content**, **detecting hate speech**, and **displaying results**. The system will be tested under normal and edge cases, including large files, **audio noise**, **non-English content**, and **high user load**.

System testing is essential for validating the system against **functional specifications** and **performance requirements**. The system will be tested to ensure that it can accurately classify **hate speech** across different content types (text, audio, video) and return results in a reasonable time frame. System tests

will also verify the **user interface** to ensure that the system is intuitive and user-friendly.

System testing also includes testing the system's **performance under stress**. The system will be subjected to a high volume of simultaneous requests to evaluate its **scalability** and ensure that it can handle the anticipated traffic and load. Performance tests will evaluate aspects such as **response time**, **latency**, and **resource consumption** to ensure that the system can handle heavy usage.

Security testing is another crucial aspect of system testing. The system will be evaluated for potential vulnerabilities, especially related to **user data** (e.g., passwords, personal information). The system must implement robust **authentication** and **authorization mechanisms** to protect user data from unauthorized access or tampering.

Finally, **compatibility testing** is performed to ensure the system works across various **devices**, **operating systems**, and **web browsers**. The hate speech detection system must be accessible to a broad audience, so testing on different devices (mobile, desktop) and browsers (Chrome, Firefox, Safari) is essential for guaranteeing that the system works for everyone.

### 7.1.4 White-Box Testing

**White-box testing** (also known as **clear-box** or **structural testing**) focuses on testing the internal workings of the system. This type of testing requires knowledge of the system's **source code** and involves testing the logic, flow, and structure of the code itself. For the hate speech detection system, **white-box testing** will be used to ensure that the **algorithmic logic** and **code implementation** are correct and efficient.

In white-box testing, every **code path**, **branch**, and **loop** in the system is tested. For example, the system's **text preprocessing** function will be tested to ensure that it correctly handles edge cases like **empty strings**, **special characters**, or **Unicode characters**. Similarly, the **model inference functions** will be tested to verify that they process inputs correctly and return accurate results.

One of the most important aspects of white-box testing for this system will be **model validation**. White-box testing will ensure that the hate speech detection model processes inputs in the intended manner, performs classification correctly, and returns the expected results. The internal **data flow** between preprocessing, model inference, and result generation will be traced and tested for correctness.

White-box testing also evaluates code for **efficiency** and **security**. For example, if the system has any critical performance bottlenecks or memory leaks, they will be identified through this testing. Security vulnerabilities, such as improper handling of **user input** or **data sanitization**, will also be evaluated in this phase.

The primary goal of white-box testing is to improve **code quality** by ensuring that the internal structure is sound and that no hidden defects exist within the system's logic.

### 7.1.5 Black-Box Testing

**Black-box testing** is the opposite of white-box testing, where the tester does not have access to the internal workings of the system. In black-box testing, the system is tested based on its **functional requirements** and **user interface**. The tester only focuses on the inputs and outputs of the system, without any knowledge of the underlying code.

For the hate speech detection system, black-box testing will evaluate how the system responds to different types of user input (text, audio, video) and whether it produces the correct classification result (i.e., **Hate Speech** or **Not Hate Speech**). The focus will be on verifying that the system functions as expected from the user's perspective.

The system will be tested with various types of data to simulate real-world usage. This includes testing with **valid inputs** (e.g., properly formatted text or clear audio) and **invalid inputs** (e.g., corrupted files, noisy audio, or mixed-language content) to ensure that the system handles errors gracefully. Test cases will also be designed to simulate **edge cases**, such as extremely long texts or very short audio clips.

Black-box testing will help identify **functional defects** that may not be visible from the developer's point of view, such as incorrect classification results, failure to handle edge cases, or improper user interactions. This testing is essential for ensuring that the system behaves as expected from the end user's perspective.

### 7.1.6 Acceptance Testing

**Acceptance testing** is the final phase of testing, conducted to determine if the system meets the **business requirements** and is ready for deployment. This type of testing is typically performed by the end users or stakeholders and focuses on validating whether the system fulfills the intended **functionalities** and **user needs**.

For the hate speech detection system, acceptance testing will involve **real users** interacting with the system to determine if it meets their expectations. This includes ensuring that the system can accurately detect hate speech across different content types and provide meaningful feedback to the user. The system's **usability**, **performance**, and **reliability** will be tested in real-world scenarios.

Acceptance testing will also validate that the system aligns with **legal and regulatory requirements**, especially those related to **data privacy**, **user rights**, and **content moderation** policies. The system must ensure that all **user data** is handled securely and complies with **GDPR** or other applicable privacy regulations.

The system will undergo **user acceptance testing (UAT)** to check if it meets the specifications outlined during the requirement gathering phase. Once the system passes UAT, it will be ready for deployment in a production environment.



## CHAPTER 8

### CONCLUSION AND FUTURE ENHANCEMENTS

#### 8.1 Conclusion

The hate speech detection system proposed in this project serves as an advanced, **multimodal solution** to tackle the growing concern of hate speech across digital platforms. The system combines state-of-the-art technologies like **BERT**, **transformer models**, and **deep learning** techniques to analyze and classify content in multiple forms—**text**, **audio**, and **video**. This integrated approach not only improves detection accuracy but also extends the system's capability to understand **context**, **tone**, and **intent**, which are critical in accurately identifying hate speech.

Throughout the development process, the system was rigorously tested using a variety of test cases and real-world scenarios. The **unit testing** of individual modules ensured that the core functionalities, such as **content preprocessing** and **model inference**, worked as expected. **Integration testing** confirmed that the system components communicated effectively, while **system testing** validated the entire workflow, ensuring the system could handle large volumes of real-time data. In particular, the ability to process multimodal content sets this system apart from existing approaches, allowing for a more comprehensive understanding of hate speech.

The project not only provides a solution for moderating hate speech but also empowers users to interact with the system through a **user-friendly interface**. The feedback and reporting features allow users to report misclassifications, contributing to the continual improvement of the system. Additionally, the system's **real-time prediction module** provides immediate feedback, enhancing

user engagement and providing a scalable solution for platforms that require constant content monitoring.

Moreover, the system's **multilingual and cross-cultural capabilities** ensure that it can be effectively deployed across diverse geographic locations and language contexts, making it a robust and versatile solution for global platforms. The **cloud-based architecture** ensures scalability, enabling the system to handle large-scale data processing without compromising on performance or response time.

In conclusion, this project provides a powerful tool for combating online hate speech. The integration of **multimodal analysis**, combined with the ability to handle large data volumes and real-time processing, positions the system as a valuable asset for both social media platforms and other digital environments where content moderation is essential. The ongoing efforts to improve the system's capabilities, coupled with the potential for further development, underscore its importance in shaping a safer and more inclusive digital space.

## 8.2 Future Enhancements

While the hate speech detection system developed in this project provides an effective solution for detecting hate speech across **text**, **audio**, and **video**, there are several opportunities for future enhancements that can make the system even more robust, scalable, and adaptable. These enhancements would improve the system's accuracy, reduce its limitations, and ensure that it keeps pace with the evolving landscape of online content and communication.

1. **Improved Detection of Sarcasm and Irony:** One of the primary challenges in hate speech detection is the accurate interpretation of **sarcasm** and **irony**. These forms of expression are particularly difficult for automated systems to detect, as they often use **contradictory language** to express the opposite of what is stated. Future enhancements could involve incorporating **emotion detection** models that analyze tone, facial expressions, and other contextual cues in both text and speech. This would help the system understand the **true intent** behind ambiguous statements and improve classification accuracy.
2. **Incorporating Multilingual and Multicultural Contexts:** The system currently supports hate speech detection in multiple languages, but there is always room for improvement in terms of understanding the cultural context in which hate speech occurs. The detection models could be further fine-tuned on **region-specific datasets**, which would help identify hate speech that may be culturally unique to certain geographical areas. This enhancement would enable the system to become even more effective in global settings where **language nuances** and **cultural norms** play a significant role in communication.

3. **Real-Time Monitoring with Feedback Loops:** While the current system provides real-time predictions, a more advanced feedback loop could be integrated to allow for continuous learning and improvement. This could involve **user feedback**, such as ratings on the accuracy of classifications, or automated flags on potentially problematic content. These inputs could then be used to **retrain** the models on an ongoing basis, ensuring that the system remains **adaptive** to new trends in language and hate speech expression. This continuous feedback would help keep the system up-to-date with the latest forms of hate speech and ensure that it improves over time.
4. **Integration with Advanced Multimedia Analysis:** The current system primarily relies on speech and text-based analysis for **video content**, but there is potential for deeper analysis by incorporating **image processing** and **computer vision** techniques. For instance, detecting hate speech in videos could be enhanced by identifying **offensive gestures** or **visual cues** that accompany the spoken word. By incorporating **advanced object detection models** like **YOLO** (You Only Look Once), the system could analyze not only the words and speech but also the visual context in videos, improving accuracy in detecting hate speech in multimedia.
5. **Context-Aware Models with Emotional Intelligence:** Future developments could explore the inclusion of **context-aware models** that analyze the **entire conversation** or the **broader context** of the content. Currently, the system analyzes individual pieces of content in isolation (e.g., a single tweet or video segment), but in some cases, hate speech may be implicit or depend on preceding or following messages. Adding **dialogue-based models** with **emotion detection** could help the system understand emotional states like **anger**, **disgust**, or **fear**, which are often

associated with hate speech.

6. **Expanding Data Sources and Datasets:** As the system grows and new trends in language evolve, it will be essential to expand the range of data sources used for training the detection models. Future enhancements could involve integrating datasets from **online forums**, **gaming platforms**, and **comment sections** on videos and articles. These datasets would provide a richer set of examples for training and fine-tuning the models, making the system more robust and capable of detecting hate speech across diverse platforms and forms of content.
7. **Enhanced User Interaction and Personalization:** The user interface could be further enhanced by integrating **personalized dashboards** for users and administrators. For example, administrators could receive detailed **analytics** about the frequency and type of hate speech on their platform, along with suggestions for **content moderation** strategies. Similarly, users could be given **feedback** on their content or interactions, with suggestions for more **inclusive language**. Such personalized features would make the system more interactive and user-centric, promoting better engagement with users.
8. **Blockchain Integration for Content Integrity:** Blockchain technology could be integrated into the system to provide **content integrity** and **authenticity** checks. Since hate speech often involves **misleading or false narratives**, blockchain could help track and verify the **source** of content and ensure that information is not manipulated. By utilizing **distributed ledger technology**, the system could offer a transparent and secure way to track the **origin** of content and **trace its evolution** through

the platform.

9. **Better Handling of Multi-Modal Data:** While the system currently handles **text**, **audio**, and **video**, future enhancements could explore even more forms of **multi-modal data**. For instance, analyzing **emoji usage** in text or evaluating **tone analysis** in text (e.g., detecting if a message is sarcastic, angry, or neutral) could provide additional insight into the **emotional intent** behind messages. This would lead to a more nuanced and accurate hate speech detection system, especially in platforms that rely heavily on informal communication styles.
10. **Cloud-Native Architecture for Scalable Deployments:** As the system grows, adopting a **cloud-native architecture** would improve its scalability and ensure seamless **global deployment**. By using platforms like **AWS**, **Google Cloud**, or **Azure**, the system could be deployed and scaled dynamically based on traffic, ensuring **low latency** and **high availability** even in regions with heavy content load. Such an architecture would allow the system to handle large-scale deployments efficiently, making it suitable for large platforms with millions of daily users.

## APPENDIX A (SOURCE CODE)

### **app.py:**

```
import streamlit as st

from streamlit import session_state

import json

import os

import whisper

import difflib

import speech_recognition as sr

from deep_translator import GoogleTranslator

from st_audiorec import st_audiorec

import moviepy.editor as mp

import torch

from transformers import AutoTokenizer ,AutoModelForSeq2SeqLM

from youtube_transcript_api import YouTubeTranscriptApi

from urllib.parse import urlparse

from pytube import YouTube

session_state = st.session_state

if "user_index" not in st.session_state:

    st.session_state["user_index"] = 0

@st.cache_resource()

def load_model_and_tokenizer(model_name):

    tokenizer = AutoTokenizer.from_pretrained(model_name)
```

```

model = AutoModelForSeq2SeqLM.from_pretrained(model_name)

return model, tokenizer

def convert_to_prompt(text):

    return f'[INST] In this task, you will be performing a
classification exercise aimed at identifying whether the given text
contains hate speech or not. Consider the text: {text} </prompt>'

def transcribe_audio_from_data(file_data):

    with open("temp.mp3", "wb") as f:

        f.write(file_data)

    model = whisper.load_model("base")

    result = model.transcribe("temp.mp3")

    os.remove("temp.mp3")

    return result["text"]

def split_text_into_chunks(text, max_chunk_size=250):

    chunks = []

    start = 0

    prompt_start = "[INST] In this task, you will be performing a
classification exercise aimed at identifying whether the given text
contains hate speech or not. Consider the text: "

    current_chunk = prompt_start

    while start < len(text):

        end = start + max_chunk_size - len(current_chunk)

        if end > len(text):

            end = len(text)

```



```

        chunk = text[start:end]

        current_chunk += chunk

        chunks.append(current_chunk)

        start = end

        current_chunk = prompt_start

    return chunks

def transcribe_video():

    try:

        model = whisper.load_model("base")

        result = model.transcribe("temp2.mp3")

        os.remove("temp2.mp3")

        return result["text"]

    except Exception as e:

        return f"Error in fetching transcript {e}"

def generate_predictions(model, tokenizer, input_texts,
max_length=128):

    device = next(model.parameters()).device

    inputs = tokenizer.batch_encode_plus(input_texts,
max_length=max_length, padding='max_length', truncation=True,
return_tensors='pt')

    input_ids = inputs['input_ids'].to(device)

    attention_mask = inputs['attention_mask'].to(device)

    with torch.no_grad():

```

```
        outputs = model.generate(input_ids=input_ids,
attention_mask=attention_mask,    max_length=max_length,    num_beams=4,
early_stopping=True)
```

```
    return tokenizer.batch_decode(outputs, skip_special_tokens=True)
```

```
def signup(json_file_path="data.json"):
```

```
    st.title("Signup Page")
```

```
    with st.form("signup_form"):
```

```
        st.write("Fill in the details below to create an account:")
```

```
        name = st.text_input("Name:")
```

```
        email = st.text_input("Email:")
```

```
        age = st.number_input("Age:", min_value=0, max_value=120)
```

```
        sex = st.radio("Sex:", ("Male", "Female", "Other"))
```

```
        password = st.text_input("Password:", type="password")
```

```
        confirm_password = st.text_input("Confirm Password:",
type="password")
```

```
    if st.form_submit_button("Signup"):
```

```
        if password == confirm_password:
```

```
            user = create_account(
```

```
                name,
```

```
                email,
```

```
                age,
```

```
                sex,
```

```
                password,
```

```

        json_file_path,

    )

    session_state["logged_in"] = True

    session_state["user_info"] = user

else:

    st.error("Passwords do not match. Please try again.")


def check_login(username, password, json_file_path="data.json"):

    try:

        with open(json_file_path, "r") as json_file:

            data = json.load(json_file)

            for user in data["users"]:

                if user["email"] == username and user["password"] ==
password:

                    session_state["logged_in"] = True

                    session_state["user_info"] = user

                    st.success("Login successful!")

                    return user

            return None

    except Exception as e:

        st.error(f"Error checking login: {e}")

        return None

```

```

def initialize_database(json_file_path="data.json"):

    try:

        if not os.path.exists(json_file_path):

            data = {"users": []}

            with open(json_file_path, "w") as json_file:

                json.dump(data, json_file)

    except Exception as e:

        print(f"Error initializing database: {e}")


def create_account(

    name,

    email,

    age,

    sex,

    password,

    json_file_path="data.json",

):

    try:

        if not os.path.exists(json_file_path) or
os.stat(json_file_path).st_size == 0:

            data = {"users": []}

```

```
else:

    with open(json_file_path, "r") as json_file:

        data = json.load(json_file)

# Append new user data to the JSON structure

user_info = {

    "name": name,

    "email": email,

    "age": age,

    "sex": sex,

    "password": password,

}

data["users"].append(user_info)

# Save the updated data to JSON

with open(json_file_path, "w") as json_file:

    json.dump(data, json_file, indent=4)

st.success("Account created successfully! You can now login.")

return user_info

except json.JSONDecodeError as e:

    st.error(f"Error decoding JSON: {e}")

return None
```

```

except Exception as e:

    st.error(f"Error creating account: {e}")

    return None

def get_transcript_from_url(url):

    try:

        url_data = urlparse(url)

        id = url_data.query[2::]

        script = YouTubeTranscriptApi.get_transcript(id)

        transcript = ""

        for text in script:

            t = text["text"]

            if t != "[Music]":

                transcript += t + " "

        return transcript

    except:

        try:

            yt = YouTube(url)

        except:

            return "Connection Error"

        stream = yt.streams.get_by_itag(251)

        stream.download("", "temp.webm")

```

```

        model = whisper.load_model("base")

        result = model.transcribe("temp.webm")

        if os.path.exists("temp.webm"):

            os.remove("temp.webm")

        return result["text"]

    except Exception as e:

        return f"Error in fetching transcript {e}"

def login(json_file_path="data.json"):

    st.title("Login Page")

    username = st.text_input("Username:")

    password = st.text_input("Password:", type="password")

    login_button = st.button("Login")

    if login_button:

        user = check_login(username, password, json_file_path)

        if user is not None:

            session_state["logged_in"] = True

            session_state["user_info"] = user

        else:

            st.error("Invalid credentials. Please try again.")

```

```
def get_user_info(email, json_file_path="data.json"):

    try:

        with open(json_file_path, "r") as json_file:

            data = json.load(json_file)

            for user in data["users"]:

                if user["email"] == email:

                    return user

            return None

    except Exception as e:

        st.error(f"Error getting user information: {e}")

        return None


def render_dashboard(user_info, json_file_path="data.json"):

    try:

        st.title(f"Welcome to the Dashboard, {user_info['name']}!")

        st.subheader("User Information:")

        st.write(f"Name: {user_info['name']}")

        st.write(f"Sex: {user_info['sex']}")

        st.write(f"Age: {user_info['age']}")

    except Exception as e:
```



```

        st.error(f"Error rendering dashboard: {e}")

def main(json_file_path="data.json"):

    st.sidebar.title("Hate Speech Detection")

    page = st.sidebar.radio(

        "Go to", ("Signup/Login", "Dashboard", "Hate Speech
Detection"), key="Hate Speech Detection",

    )

    if page == "Signup/Login":

        st.title("Signup/Login Page")

        login_or_signup = st.radio("Select an option", ("Login",
"Signup"), key="login_signup")

        if login_or_signup == "Login":

            login(json_file_path)

        else:

            signup(json_file_path)

    elif page == "Dashboard":

        if session_state.get("logged_in"):

            render_dashboard(session_state["user_info"])

        else:

            st.warning("Please login/signup to view the dashboard.")

    elif page == "Hate Speech Detection":

```

```

if session_state.get("logged_in"):

    user_info = session_state["user_info"]

    model_folder= "hatespeech"

    model, tokenizer = load_model_and_tokenizer(model_folder)

    st.title("Hate Speech Detection")

    media_format = st.radio("Choose media format", ("Audio",
"Video", "URL"))

    if media_format == "Audio":

        options = ["Record", "Upload"]

        choice = st.radio("Choose an option", options)

        if choice == "Record":

            recognizer = sr.Recognizer()

            microphone = sr.Microphone()

            if st.button("START RECORDING"):

                with microphone as source:

                    st.info("Listening...")

                    recognizer.adjust_for_ambient_noise(source)

                    audio = recognizer.listen(source)

                    try:

                                                                transcript =
recognizer.recognize_google(audio, language="en")

```

```

        st.write(f"Transcription: {transcript}")

    except sr.UnknownValueError:

        st.write("Could not understand the audio.
Please try again.")

        return

    except sr.RequestError as e:

        st.write("Could not understand the audio.
Please try again.")

        return

    if transcript:

        chunks = split_text_into_chunks(transcript)

        is_hate_speech = False

        for chunk in chunks:

            prompt = convert_to_prompt(chunk)

            predictions =
generate_predictions(model, tokenizer, [prompt])

            if predictions[0] == "Hate Speech":

                is_hate_speech = True

                break

        if is_hate_speech:

            overall_prediction = "Hate Speech"

            st.markdown(f'<p style="color:red;
font-size:20px;">{overall_prediction}</p>', unsafe_allow_html=True)

```

```

else:

    overall_prediction = "Not Hate Speech"

    st.markdown(f'<p style="color:green;
font-size:20px;">{overall_prediction}</p>', unsafe_allow_html=True)

elif choice == "Upload":

    st.write("Upload an audio file:")

    audio = st.file_uploader("Upload an audio file",
type=["mp3", "wav", "ogg"])

    if audio is not None:

        st.audio(audio, format="audio/wav")

transcription =
transcribe_audio_from_data(audio.read()).upper()

    if transcription:

chunks =
split_text_into_chunks(transcription)

is_hate_speech = False

for chunk in chunks:

    prompt = convert_to_prompt(chunk)

predictions =
generate_predictions(model, tokenizer, [prompt])

    if predictions[0] == "Hate Speech":

        is_hate_speech = True

        break

```

```

        if is_hate_speech:

            overall_prediction = "Hate Speech"

            st.markdown(f'<p style="color:red;
font-size:20px;">{overall_prediction}</p>', unsafe_allow_html=True)

        else:

            overall_prediction = "Not Hate Speech"

            st.markdown(f'<p style="color:green;
font-size:20px;">{overall_prediction}</p>', unsafe_allow_html=True)

    elif media_format == "Video":

        video = st.file_uploader("Upload Video", type=["mp4",
"avi", "mov", "wmv"])

        if video is not None:

            with open("temp.mp4", "wb") as f:

                f.write(video.read())

            video1 = mp.VideoFileClip('temp.mp4')

            with st.spinner("Transcribing video..."):

                audio_file = video1.audio

                audio = audio_file.write_audiofile('temp2.mp3')

            transcription = transcribe_video()

            if transcription:

                chunks =
split_text_into_chunks(transcription)

                is_hate_speech = False

```

```

        for chunk in chunks:

            prompt = convert_to_prompt(chunk)

            predictions =
generate_predictions(model, tokenizer, [prompt])

            if predictions[0] == "Hate Speech":

                is_hate_speech = True

                break

        if is_hate_speech:

            overall_prediction = "Hate Speech"

            st.markdown(f'<p style="color:red;
font-size:20px;">{overall_prediction}</p>', unsafe_allow_html=True)

        else:

            overall_prediction = "Not Hate Speech"

            st.markdown(f'<p style="color:green;
font-size:20px;">{overall_prediction}</p>', unsafe_allow_html=True)

    elif media_format == "URL":

        url = st.text_input("Enter the video URL:")

        if url:

            transcript = get_transcript_from_url(url)

            if transcript:

                chunks = split_text_into_chunks(transcript)

```

```

is_hate_speech = False

for chunk in chunks:

    prompt = convert_to_prompt(chunk)

    predictions = generate_predictions(model,
tokenizer, [prompt])

    if predictions[0] == "Hate Speech":

        is_hate_speech = True

        break

if is_hate_speech:

    overall_prediction = "Hate Speech"

    st.markdown(f'<p style="color:red;
font-size:20px;">{overall_prediction}</p>', unsafe_allow_html=True)

else:

    overall_prediction = "Not Hate Speech"

    st.markdown(f'<p style="color:green;
font-size:20px;">{overall_prediction}</p>', unsafe_allow_html=True)

else:

    st.error("Error fetching transcript from the
URL.")

else:

    st.warning("Please login/signup to use the hate speech
detection feature.")

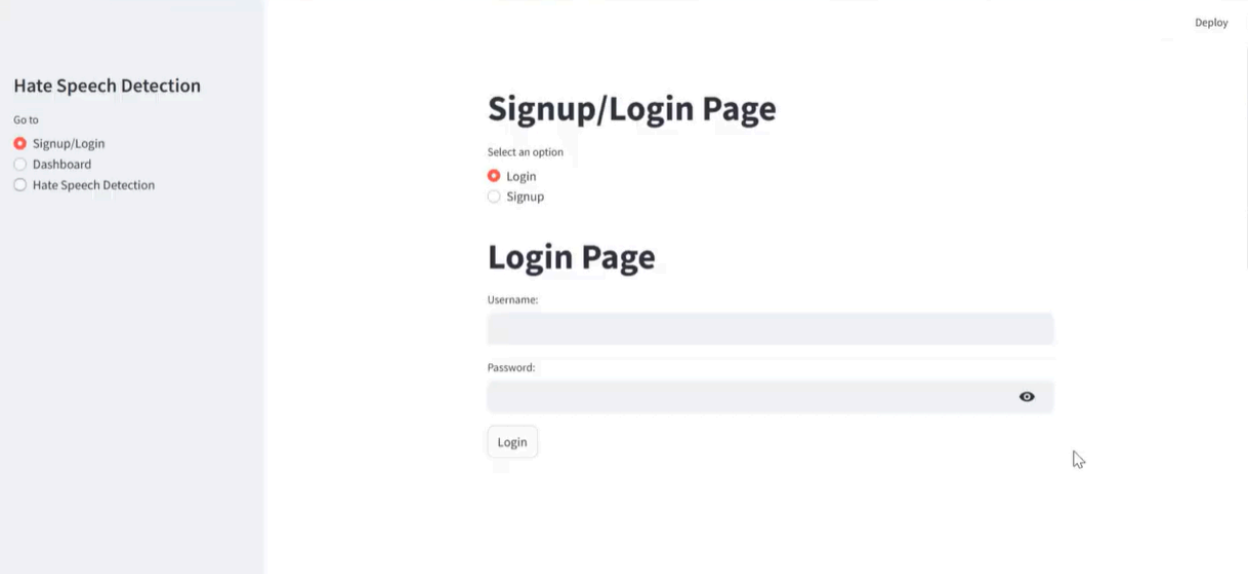
if __name__ == "__main__":

    initialize_database()

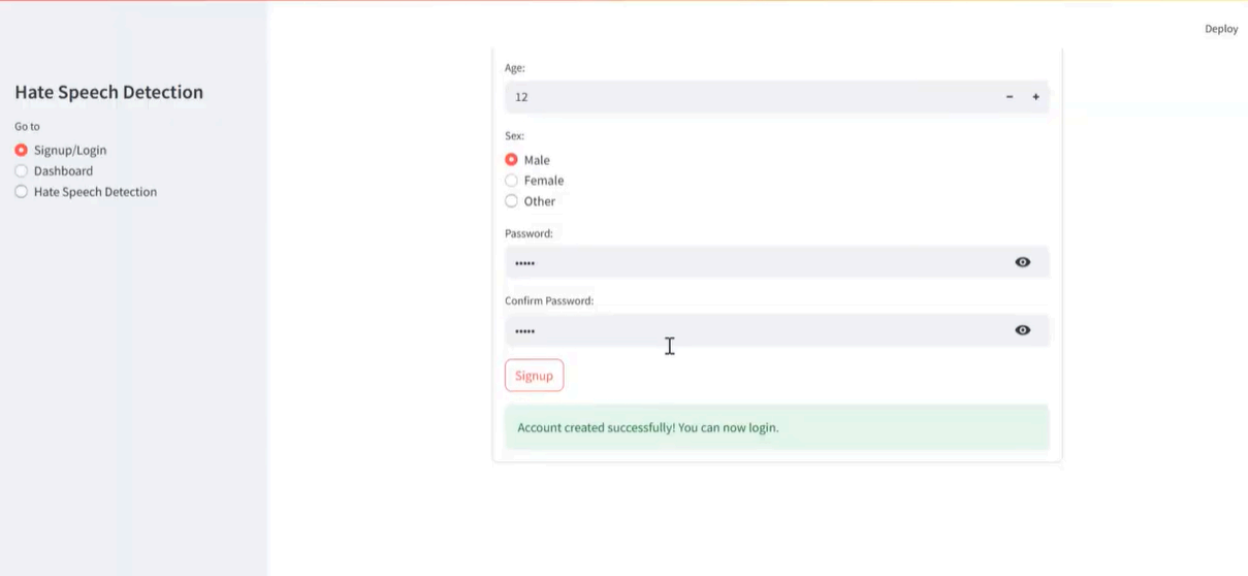
```

```
main()
```

## APPENDIX B (SCREENSHOTS)

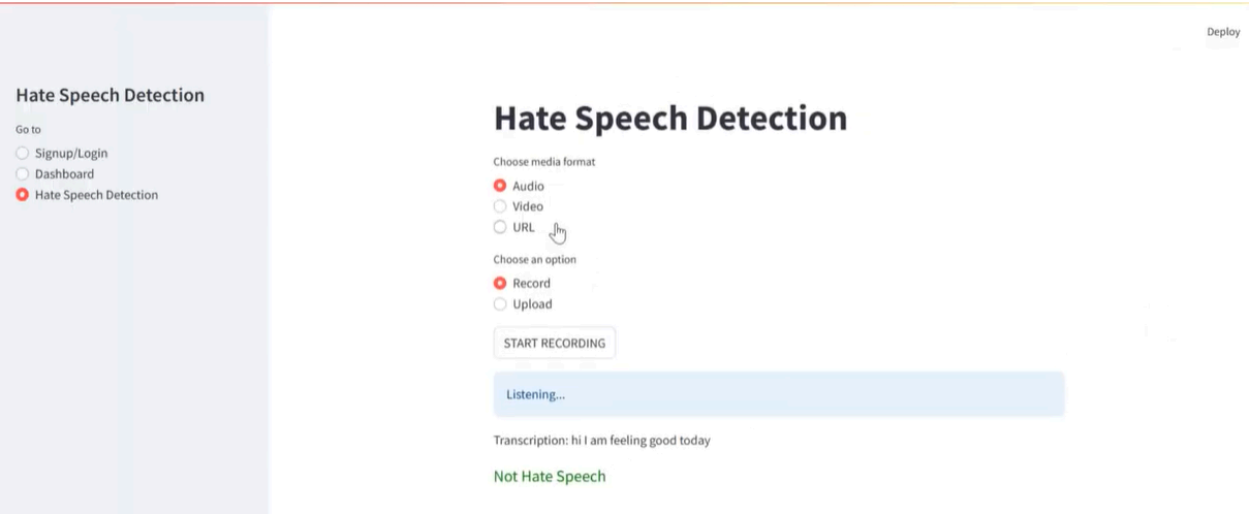
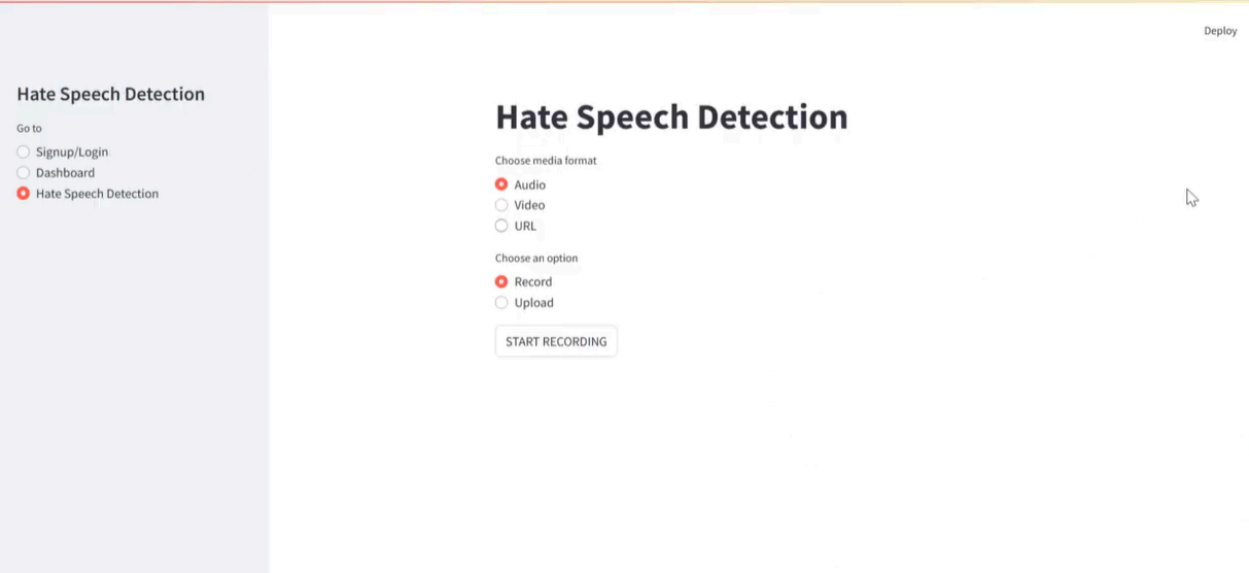
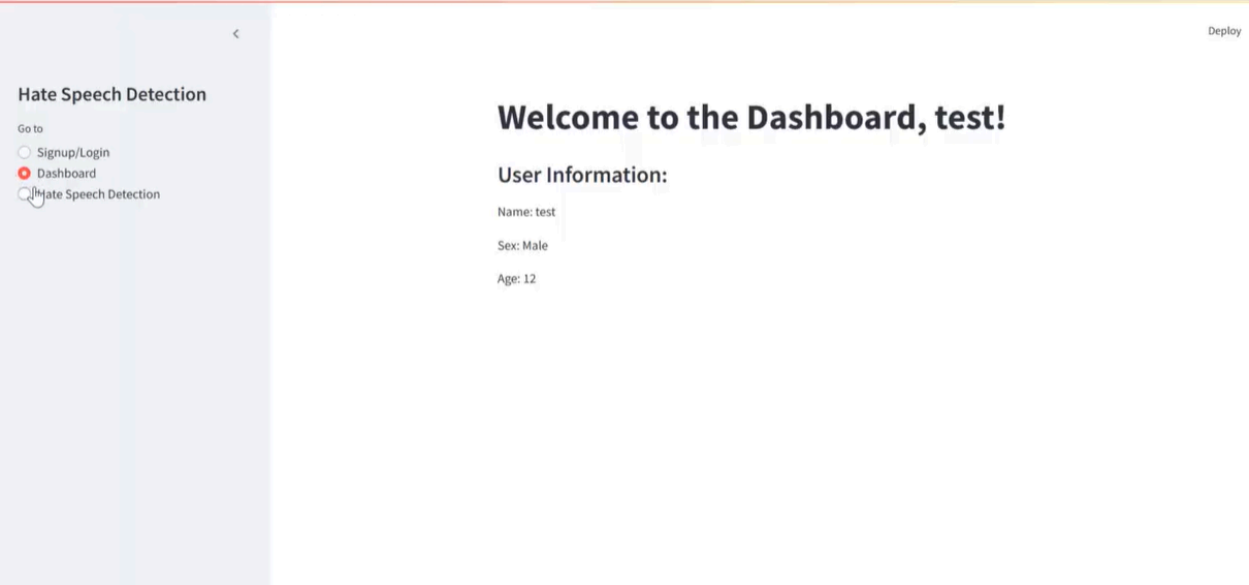


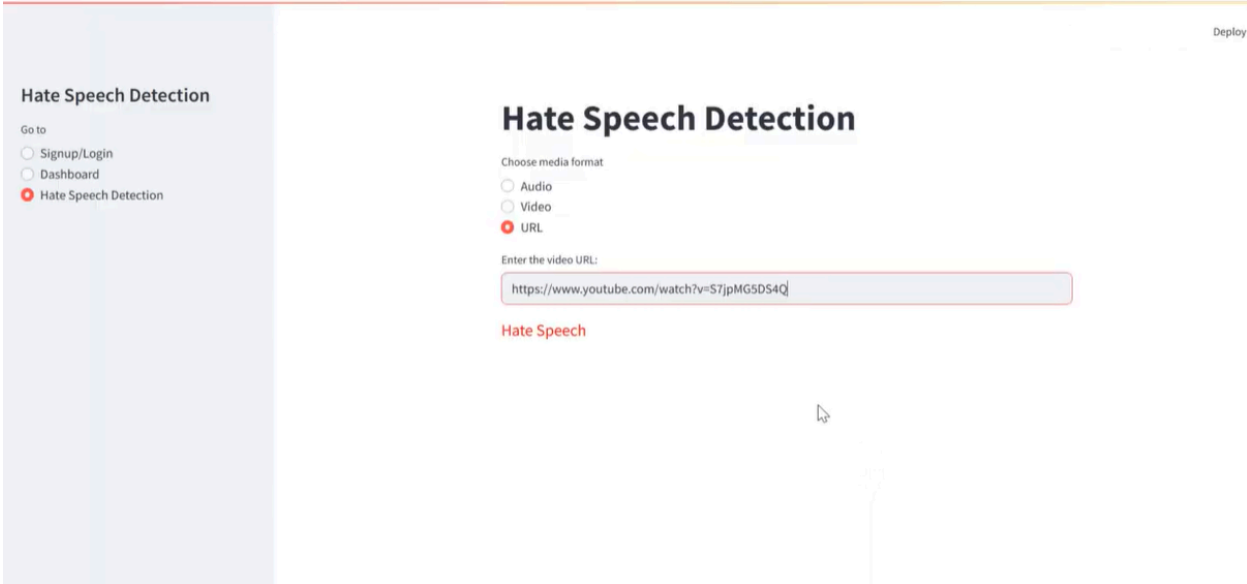
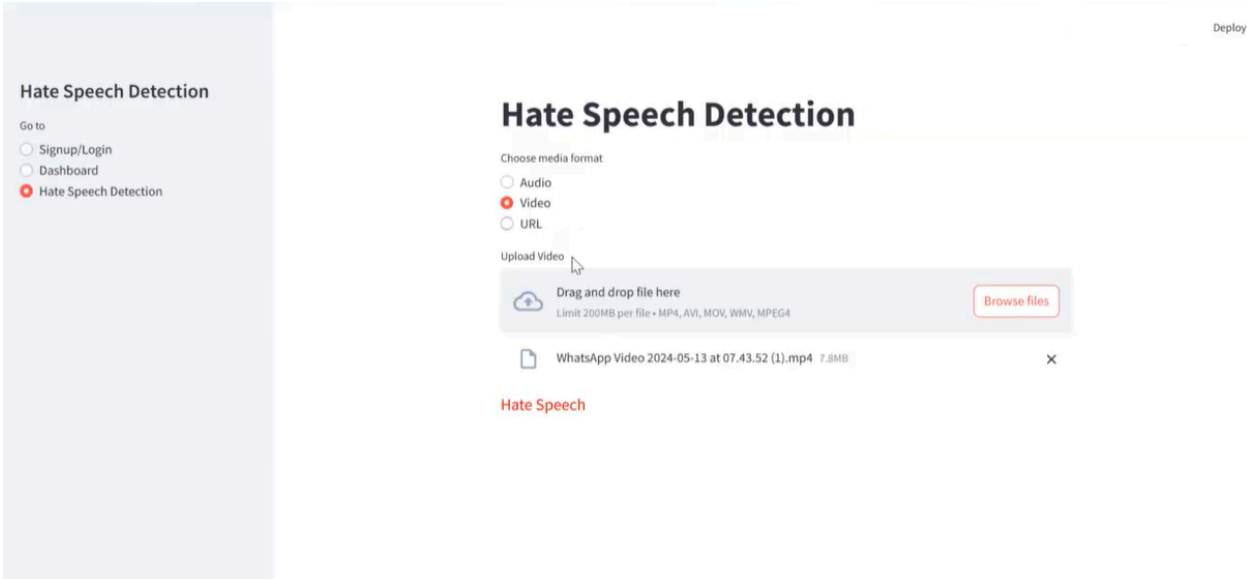
This screenshot shows the 'Signup/Login Page' of a web application. On the left, a sidebar titled 'Hate Speech Detection' contains a 'Go to' section with three radio buttons: 'Signup/Login' (selected), 'Dashboard', and 'Hate Speech Detection'. The main content area has a title 'Signup/Login Page' and a 'Select an option' section with 'Login' (selected) and 'Signup' radio buttons. Below this is a 'Login Page' section with a 'Username:' label and a text input field, a 'Password:' label and a text input field with a toggle icon, and a 'Login' button. A 'Deploy' link is visible in the top right corner.



This screenshot shows the 'Signup' page of the web application. The sidebar on the left is identical to the previous screenshot. The main content area features a 'Signup' form with the following fields: 'Age:' with a numeric input containing '12', 'Sex:' with radio buttons for 'Male' (selected), 'Female', and 'Other', 'Password:' with a text input containing '\*\*\*\*' and a toggle icon, and 'Confirm Password:' with a text input containing '\*\*\*\*' and a toggle icon. A 'Signup' button is located below the form. A green success message at the bottom of the form states 'Account created successfully! You can now login.' A 'Deploy' link is visible in the top right corner.







## REFERENCES

1. Sasidaran, K., & Geetha, J. (2024, August). Multimodal Hate Speech Detection using Fine-Tuned Llama 2 Model. In *2024 International Conference on Intelligent Algorithms for Computational Intelligence Systems (IACIS)* (pp. 1-6). IEEE.
2. Hee, M. S., Sharma, S., Cao, R., Nandi, P., Chakraborty, T., & Lee, R. K. W. (2024). Recent advances in hate speech moderation: Multimodality and the role of large models. *arXiv preprint arXiv:2401.16727*.
3. Faria, F. T. J., Baniata, L. H., & Kang, S. (2024). Investigating the Predominance of Large Language Models in Low-Resource Bangla Language Over Transformer Models for Hate Speech Detection: A Comparative Analysis.
4. Malode, V. M. (2024). *Benchmarking public large language model* (Doctoral dissertation, Technische Hochschule Ingolstadt).
5. Dehghani, M. (2024). A comprehensive cross-language framework for harmful content detection with the aid of sentiment analysis. *arXiv preprint arXiv:2403.01270*.
6. Chun, J. (2024). MultiSentimentArcs: a novel method to measure coherence in multimodal sentiment analysis for long-form narratives in film. *Frontiers in Computer Science*, 6, 1444549.
7. Hee, M. S., Singh, K., Si Min, C. N., Choo, K. T. W., & Lee, R. K. W. (2024, May). Brinjal: A Web-Plugin for Collaborative Hate Speech Detection. In *Companion Proceedings of the ACM on Web Conference 2024* (pp. 1063-1066).
8. Wang, J., Jiang, H., Liu, Y., Ma, C., Zhang, X., Pan, Y., ... & Zhang, S. (2024). A comprehensive review of multimodal large language models: Performance and challenges across different tasks. *arXiv preprint arXiv:2408.01319*.

9. Hadi, M. U., Al Tashi, Q., Shah, A., Qureshi, R., Muneer, A., Irfan, M., ... & Shah, M. (2024). Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects. *Authorea Preprints*.
10. Sharma, A., Gupta, A., & Bilalpur, M. (2023). Argumentative Stance Prediction: An Exploratory Study on Multimodality and Few-Shot Learning. *arXiv preprint arXiv:2310.07093*.