College of Computing and Informatics

Computer Science Department

University of Sharjah

Fall 2024/2025

Senior Project Final Report:

Automated Video Content Appropriateness Evaluator for Schools

A project submitted

in partial fulfilment of the requirements for the degree of

Bachelor in Computer Science

By

Kavisna Uma Kandan (U21103479)

Athifa Nizamudeen (U21104061)

Sana Aziz (U21102470)

Supervised By

Dr. Mohammad Abdullah A.R. Alsmirat

December 2024

ACKNOWLEDGEMENT

We would like to highly acknowledge our supervisor, Dr. Mohammad Abdullah A.R. Alsmirat (Associate Professor in Computer Science Department, University of Sharjah) for his continuous valuable support and guidance throughout this project. In addition, the school teachers who participated in the survey are sincerely thanked for allowing us to collect the necessary data crucial for laying the foundation of this project. Finally, we extend huge gratitude to our family for always supporting and encouraging us to do our best.

UNDERTAKING

This is to declare that the project entitled “Automated Video Content Appropriateness Evaluator for Schools” is an original work done by undersigned, in partial fulfillment of the requirements for the degree “Bachelor in Computer Science” at the Computer Science Department, College of Computing and Informatics, University of Sharjah, UAE. All the analysis, design, and system development have been accomplished by the undersigned. Moreover, this project has not been submitted to any other college or university.

Kavisna Uma Kandan *Kavisna*

Athifa Nizamudeen *Athifa*

Sana Aziz *Sana*

ABSTRACT

The evaluation of video content suitability for students is a critical task for teachers but can be time-consuming. To address this challenge, we propose the development of the Appropriate Content Evaluator for Schools (ACES), a web application designed to automate the evaluation process. By leveraging computer vision, Natural Language Processing (NLP), machine learning and deep learning techniques, ACES aims to assess video content based on criteria such as Presentation Complexity and Language Complexity. Among various machine learning models tested, Random Forest showed the best performance in assessing Presentation Complexity. As for Language Complexity, DistilBERT-BiLSTM model showed potential in predicting the grade level based on the video's transcript extracted using OpenAI Whisper. Overall, our approach assists teachers and enhances the learning experience for students.

Table of Contents

List of Figures

List of Tables

CHAPTER 1: Introduction

1.1 Overview

Our project focuses on developing a solution called Appropriate Content Evaluator for Schools (ACES), to assist school teachers efficiently choose suitable and quality videos for their students. ACES is a web application that will employ machine learning and deep learning models to automatically assess the appropriateness of videos based on several criteria. The platform lets the user upload a video and customize the appropriateness evaluation criteria according to their preferences. We identified the essential criteria for assessing video appropriateness from a teacher's viewpoint through a survey. The criteria include Presentation Complexity, Language Complexity, Mature Content, Clarity of Image and Sound, and Inclusion of Real-Life Examples. The models will be trained on large datasets to detect and classify various elements within videos based on visual and auditory patterns. We will leverage techniques such as computer vision, Natural Language Processing (NLP), image processing, video processing, and audio processing to implement the models, and HTML, CSS, JavaScript, and Flask to design the web application. Our proposed solution will be built upon the stages in the waterfall software development life cycle model.

1.2 Problem and Motivation

Education is one of the many fields that has revolutionized significantly ever since the advancement of technology. One such prominent change is the use of online resources of all types including texts, audios, and videos for learning and teaching [1]. In teaching particularly, videos have become the go-to visualization tool for teachers to pique students' interest for better understanding and to provide an interactive learning session [1], [2]. However, a vast variety of videos for every topic are available on the internet and choosing the suitable one to use in lessons has become a tough task to accomplish. According to research by the Department for Education in the United Kingdom, 48% of the teachers answered that lesson planning is a time-consuming job [2]. On top of that, due to the ubiquity of the internet, everyone has the freedom to share and access anything online, and no restrictions are imposed on the appropriateness of the uploaded content. This further increases the difficulty of the video selection process for teachers [1], [3].

Computer vision and NLP are ever-growing fields of Artificial Intelligence that have found their use in various areas including education. Machine learning and deep learning are predominantly used in both the fields to train computers to make decisions on their own. Computer vision enables computers to understand visual patterns from images and videos, whereas NLP allows computers to recognize human language in any form including audio and text [4], [5]. Since our work involves analysing both the scenes and language in a video, the promising capability shown by the two fields paves a way for automated video classification in an efficient manner. In this project, we will integrate computer vision, NLP, and web application development to build ACES for automatic evaluation of videos' appropriateness.

1.3 Aim and Objectives

Our project aims to deliver a web application for school teachers called ACES, that can accurately analyze uploaded videos to determine appropriateness based on user-defined criteria. Teachers should be able to upload video file or link, select criteria, view video suitability results, and give feedback on the results easily. Below are the objectives we have defined to achieve our goal:

1. Collect necessary information related to video usage from teachers who are the end-users of the system
2. Develop skills and knowledge on the techniques that will be used in this project
3. Construct implementation plan for each criterion including the dataset and workflow
4. Design a user-friendly interface of the web application
5. Implement ACES and test its accuracy, functionality, and usability

1.4 Project Environment

The languages we have chosen to use are Python for model implementation and backend web development, and HTML, CSS, and JavaScript for frontend web development. The high-level software requirements include a MySQL database system, and different programming environments and frameworks. The uploaded videos' metadata and their corresponding appropriateness results and user feedback will be stored in the database for model learning to improve the accuracy of the future results. The programming environments are Jupyter Notebook to develop machine learning models; Google Colab to leverage its GPU service for developing deep learning models that require high computational power; and Visual

Studio Code and PyCharm to design the web application. The main libraries and frameworks are scikit-learn and TensorFlow for model implementation, and Flask for backend web development.

1.5 Limitations and Risks

The foremost issue that needs to be addressed is about the appropriateness evaluation criteria used for video content analysis. This list of criteria will be formed based on the teachers' responses in the survey. However, every individual will have a different point of view regarding the factors affecting the videos' appropriateness and there are no correct answers to this. Since it is impractical to include all the factors mentioned in the survey results, we have decided to limit the list to a certain number by picking out the ones selected by most teachers.

As for datasets, we could not find a suitable one with videos and required features for the Presentation Complexity criterion. Hence, we created our own dataset using manually collected videos from YouTube. However, we were unable to find experts, such as teachers and video makers, to help annotate our self-created datasets. Due to this, all the manually collected videos were annotated by the three members of this project based on past school knowledge and experience. To minimize any impact on model performance, we took measures such as independent annotations by three people and multiple verifications, ensuring unbiased, consistent, and reliable labelling.

In addition, we initially relied on a Python library called pytubefix to automatically download the videos from YouTube. This library along with other video downloading libraries stopped working at some point due to changes in the YouTube's processing script. Hence, we manually downloaded videos for model testing and improvement using an online website. We also removed the video URL input feature on the front-end as there was no automated solution for downloading the videos in the back-end.

1.6 Project Expected Output

The project aims to deliver the following:

- Implementation plan for each criterion describing the dataset and workflow
- User interface design for the web application
- Web application called ACES that will operate using machine learning and deep learning
- Database implementation to store uploaded videos' details and their appropriateness results and user feedback on those results for model learning purpose

1.7 Project Timeline

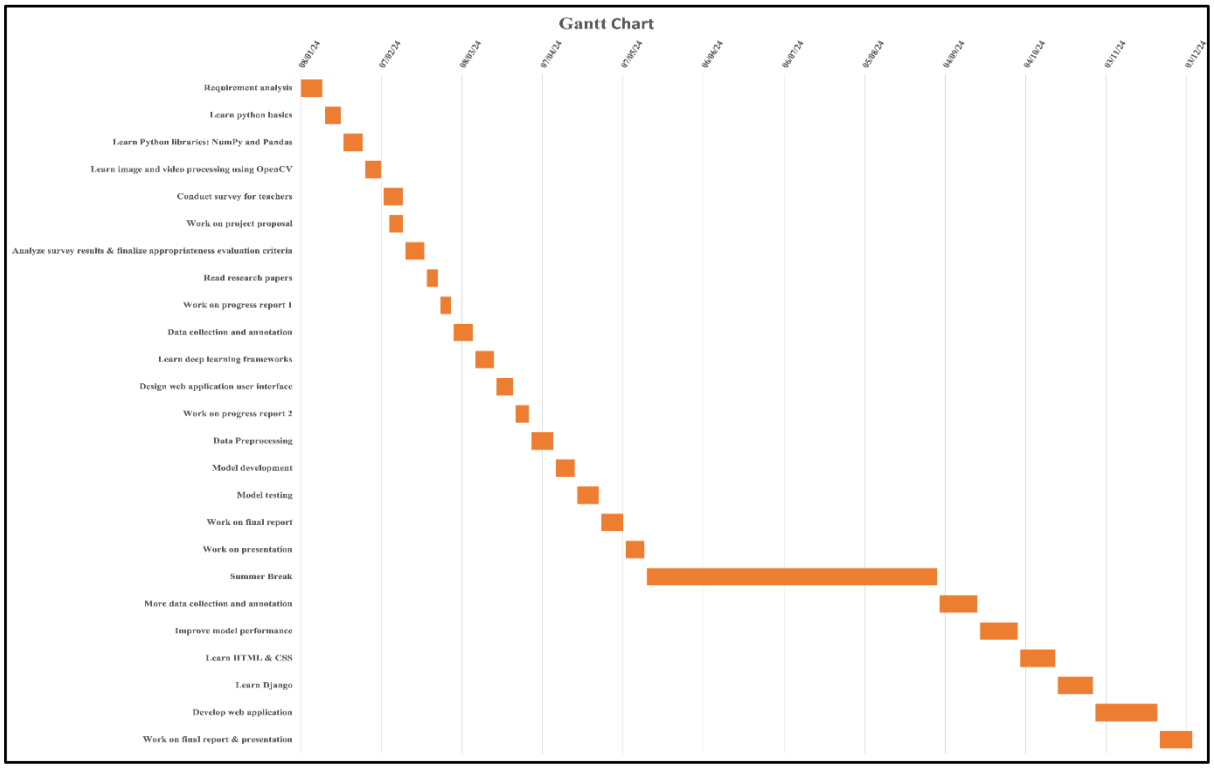

Figure 1-1: Gantt chart for visualizing the project timeline and the dependencies between each task

Table 1-1: Project timeline showcasing the expected durations to complete the required tasks

Tasks	Start Date	End Date	Number of days to Complete
Junior Project			
Requirement analysis	08/01/2024	16/01/2024	8
Learn python basics	17/01/2024	23/01/2024	6
Learn Python libraries: NumPy and Pandas	24/01/2024	31/01/2024	7
Learn image and video processing using OpenCV	01/02/2024	07/02/2024	6
Conduct survey for teachers	08/02/2024	15/02/2024	7
Work on project proposal	10/02/2024	15/02/2024	5
Analyze survey results & finalize appropriateness evaluation criteria	16/02/2024	23/02/2024	7
Read research papers	24/02/2024	28/02/2024	4
Work on progress report 1	29/02/2024	04/03/2024	4
Data collection and annotation	05/03/2024	12/03/2024	7
Learn deep learning frameworks	13/03/2024	20/03/2024	7
Design web application user interface	21/03/2024	27/03/2024	6
Work on progress report 2	28/03/2024	02/04/2024	5
Data Preprocessing	03/04/2024	11/04/2024	8
Model development	12/04/2024	19/04/2024	7
Model testing	20/04/2024	28/04/2024	8
Work on final report	29/04/2024	07/05/2024	8
Work on presentation	08/05/2024	15/05/2024	7
Summer Break	16/05/2024	01/09/2024	108
Senior Project			
More data collection and annotation	02/09/2024	16/09/2024	14
Improve model performance	17/09/2024	01/10/2024	14
Learn HTML & CSS	02/10/2024	15/10/2024	13
Learn Django	16/10/2024	29/10/2024	13
Develop web application	30/10/2024	22/11/2024	23
Work on final report & presentation	23/11/2024	05/12/2024	12

CHAPTER 2: Literature Review

To this day, there is no work on determining videos' appropriateness that is both a real-time system and dedicated to school teachers. There exists many computer vision-based image and video content moderation services in the form of APIs such as [6], and [7], specifically designed for developers to integrate into their applications. However, educators do not have easy access to them to check for the presence of mature content as either programming background is necessary, payment is required, or a restriction is set on the number of allowed usage. On top of that, the implementation details of these services are not explicitly explained, hence they cannot be referred to in this project.

On the other hand, a research paper and a few web applications following different methodologies are available. One of them [3] involves checking if the YouTube videos played on Smart TV are suitable or not for the viewers. This was done by doing real-time comparison of the audiences' age with the age-group label given to the viewing video. Computer vision was used to identify the age of the audiences through facial recognition. Then, Random Forest algorithm was employed to classify the videos into three different age-groups (adult, teenager, child) using the videos' metadata. A drawback of this method is that it does not verify whether the content of the video matches the metadata.

Safe Search Kids [8] is a web application that allows teachers and parents to search for appropriate videos for children using the safe video search engine. They assess the quality of the videos from different sources before making them available on their platform. Another website called Common Sense [9] recommends age-appropriate videos for both entertainment and educational purposes. Under the education section, they provide videos tailored to meet the needs of students in different grade levels. Both websites offer suitable videos on various subjects. But, unlike the proposed solution, users cannot upload a video of their choice as real-time checking is not supported. In addition, the videos on these websites are manually evaluated and labelled as either appropriate or inappropriate and this is inefficient for huge amounts of videos.

An important criteria for assessing video appropriateness for school students is linguistic complexity. Various methods using machine learning and deep learning have been proposed by researchers. [10] used Light Gradient Boost Method (LGBM), a machine learning

model to analyze text complexity based on handcrafted-linguistic features computed from readability formulas. However, Long Short-Term Memory (LSTM), a deep neural network that does not rely on manual feature engineering showed a much better performance with an accuracy of around 98% in [11]. In another study as well [12], Recurrent Neural Network (RNN) architecture was utilized, achieving an accuracy slightly higher than the machine learning models of the same study and [10]. This demonstrates that deep learning approaches are more suitable for determining language complexity.

Overall, existing solutions fail to provide real-time, user-focused, and comprehensive evaluation of videos' appropriateness. Our proposed solution, called ACES, utilizes machine learning and deep learning to automate the video appropriateness evaluation process based on several user-selected criteria, enabling teachers to effectively and efficiently choose suitable videos for their students.

CHAPTER 3: Requirement Engineering and Analysis

3.1 Survey Analysis

We conducted an online survey among school teachers, who are the end-users of the proposed automated video appropriateness checking system. A total of 27 teachers from different schools participated in this survey and their feedback helped us ensure the software being created is satisfactory.

The purpose of this survey is outlined as follows:

- To study the significance of our solution to teachers
- To design the appropriateness evaluation criteria
- To identify the requirements of our software based on teachers' needs

Survey link: https://forms.gle/KCNqDnnYqX5ZuPqQ7

3.1.1 Respondents Demographic Overview

The responses in the survey came from different age groups of adults (young, middle-aged, old), and this is beneficial in developing an unbiased system. Most of the respondents are teachers to kindergarten and high-school students for various subjects. Some are teaching general subjects such as Mathematics, Science, English, Information Technology, Physical

Education, and Health Education, while others are teaching vocational subjects related to Engineering, Aviation, and Computer Science. The type of videos shown in each area of study may largely vary due to differences in the required explanations. This indicates that diverse opinions were obtained regarding the suitability of these videos.

3.1.2 Key Findings

Most of the teachers use videos frequently in their lessons (Figure 3-1) and have also agreed that the videos' content will impact students' growth to some extent (Figure 3-2). This clearly shows the significance of selecting appropriate videos for educational purposes. However, many teachers watch the video completely to conclude whether it is appropriate or not before showing it to the students (Figure 3-4). To avoid this time-consuming process, the automated video appropriateness checker solution was proposed to which around 93% of the teachers gave their approval (Figure 3-6).

As the appropriateness evaluation criteria is an integral part of the proposed system, we requested the teachers to choose the factors that will most likely affect the videos' appropriateness. The options given in the survey (Figure 3-3) were formed after an in-depth group discussion in which the criteria set of [1] served as a guideline. In addition, many of the teachers prefer these criteria to be customizable (Figure 3-7). Lastly, the videos that will be used for testing as well as dataset construction if needed will be taken from YouTube as this is the most accessed platform by the teachers (Figure 3-5).

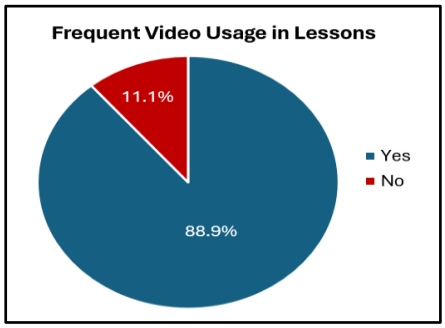

Figure 3-1: Proportion of teachers using videos frequently in lessons

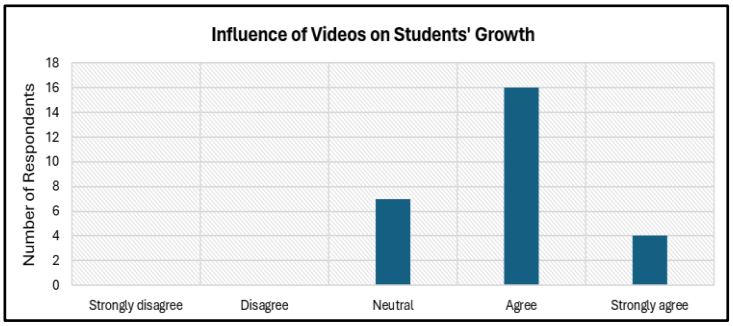

Figure 3-2: Teachers' view on the influence of videos on students' growth

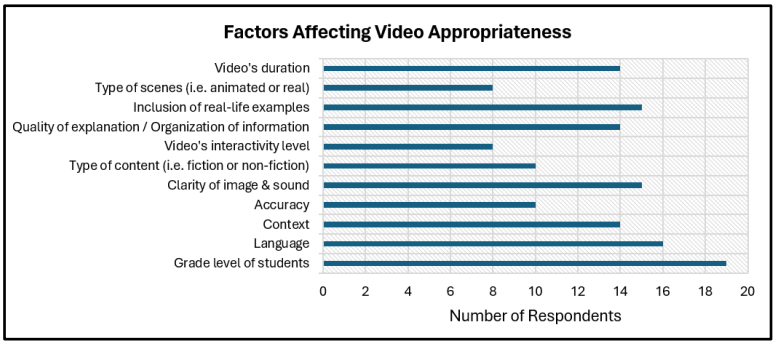

Figure 3-3: Factors considered to be essential for video appropriateness analysis according to teachers

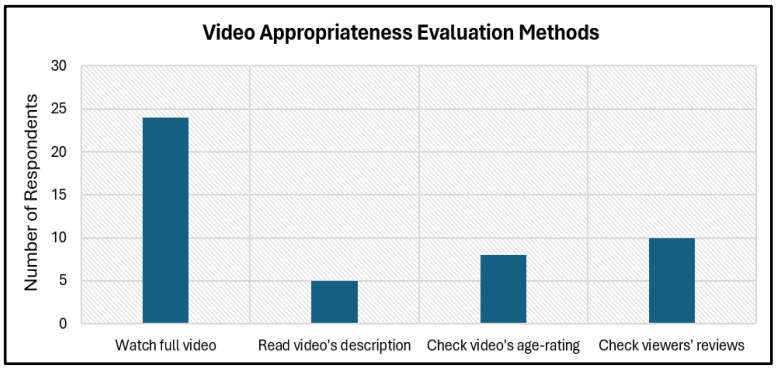

Figure 3-4: Current video appropriateness evaluation methods followed by teachers

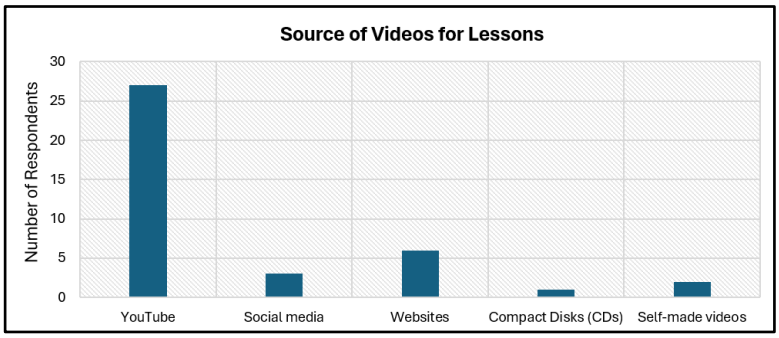

Figure 3-5: Teachers' most used video sources for lessons

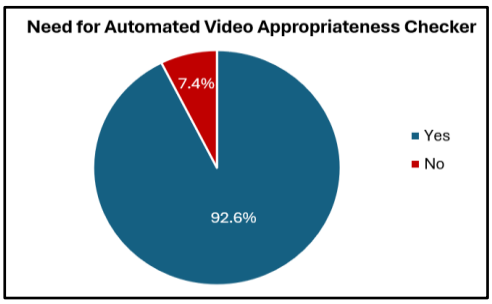

Figure 3-6: Teachers' view on the development of automated system for video appropriateness evaluation

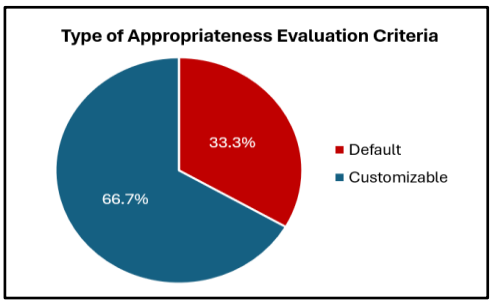

Figure 3-7: Teachers' preference on the type of appropriateness evaluation criteria in the automated system

3.2 Appropriateness Evaluation Criteria

Based on the insights gained from the analysis of the survey results in Figure 3-3, we have defined five criteria in Table 3-1 to evaluate the appropriateness of a video. This selection was made by taking into account the number of survey responses, relevance, and feasibility, where higher outcome was preferred for all.

Table 3-1: Appropriateness evaluation criteria for video content analysis

Criteria	Definition
Presentation Complexity	Complexity of the video's presentation style
Language Complexity	Complexity of the language used in the video
Mature Content	Presence of vulgar language, indecent behaviors or habits, drugs, violence, sex, nudity [13]
Clarity of Image and Sound	Use of high quality images and sound for pleasant viewing
Inclusion of Real-Life Examples	Incorporation of actual examples in explanations for students to relate and understand better

3.3 Functional and Non-Functional Requirements

Table 3-2: Functional requirements for the system and users

<table>
<tr><th colspan="2">Functional Requirements</th></tr>
<tr><th>System</th><th>Users</th></tr>
<tr><td rowspan="2">• Extract frames and audio from the uploaded video

• Classify various elements within the uploaded video based on user-selected criteria

• Output whether video is appropriate or not after classification

• Store uploaded videos and their appropriateness results and user feedback on the results in the database

• Execute model update request from admin using data from the database</td><td>1. Teacher
• Upload a video's link or a local video file with valid file format

• Select one or more appropriateness criteria for video analysis

• Provide feedback on the appropriateness result</td></tr>
<tr><td>2. Admin
• Check models' accuracy

• Update models that run the software

• Check teachers' activity</td></tr>
</table>

Table 3-3: Non-functional requirements for the system

Non-Functional Requirements
• Web application must provide simple and intuitive UI where minimal number of steps are required for the user to upload videos and view result. • System must support fast video upload. • System must respond to user queries in minimal time. • System must provide accurate and reliable video content analysis. • System must provide secure storage of uploaded videos and feedback on the appropriateness results for users. • Web application should be compatible with commonly used web browsers. • System should be able to handle multiple concurrent uploads and analyses. • Documentation on the models' implementation should be written clearly to perform maintenance and testing easily.

3.4 Use Case Diagram

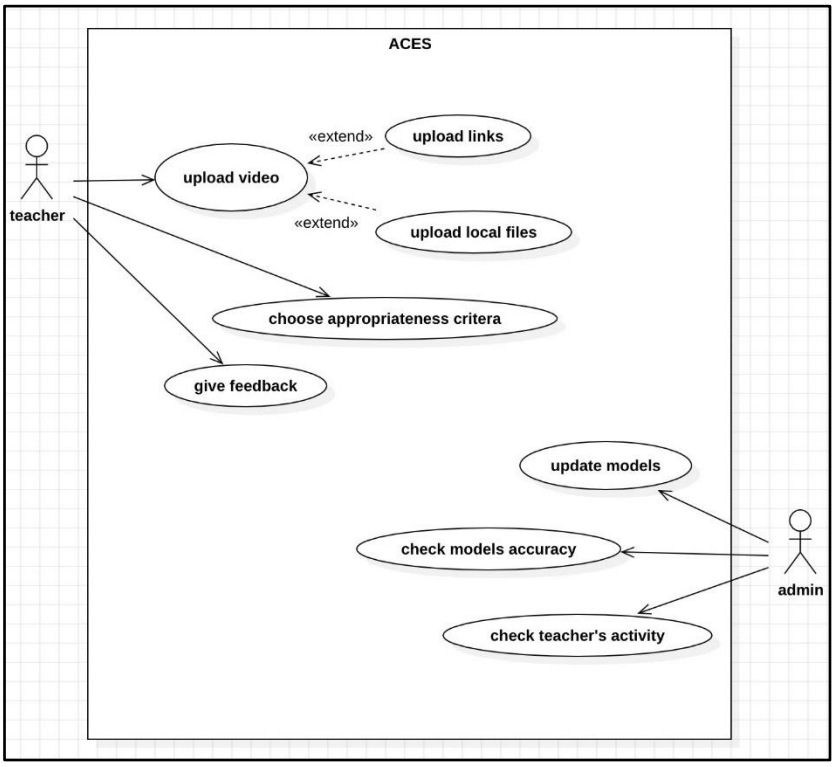

Figure 3-6: Use case diagram for visualizing the functional requirements of users

CHAPTER 4: Architecture and Design

4.1 Software Architecture

Our software requires client-server architecture where the client interacts with the web application through a browser interface, and the server handles processing tasks such as video appropriateness analysis. The components of the software on the server-side that perform video processing are representative of the appropriateness evaluation criteria, and they include Presentation Complexity, Language Complexity, Mature Content, Clarity of Image and Sound, and Inclusion of Real-Life Examples (Table 3-1). Mature Content consists of two parts which assess the decency of the language and scenes in the video. When the client requests the server to evaluate the appropriateness of the uploaded video through the web application interface, the server will forward the request to the correct components based on the user-selected criteria. The results produced by the components will be sent as a response from the server to the client. In addition, the data in the database system can be accessed by the server to re-train the models (video processing components) for better performance. An overview of the software architecture is shown in Figure 4-1 where the client-server interaction and the major components have been illustrated.

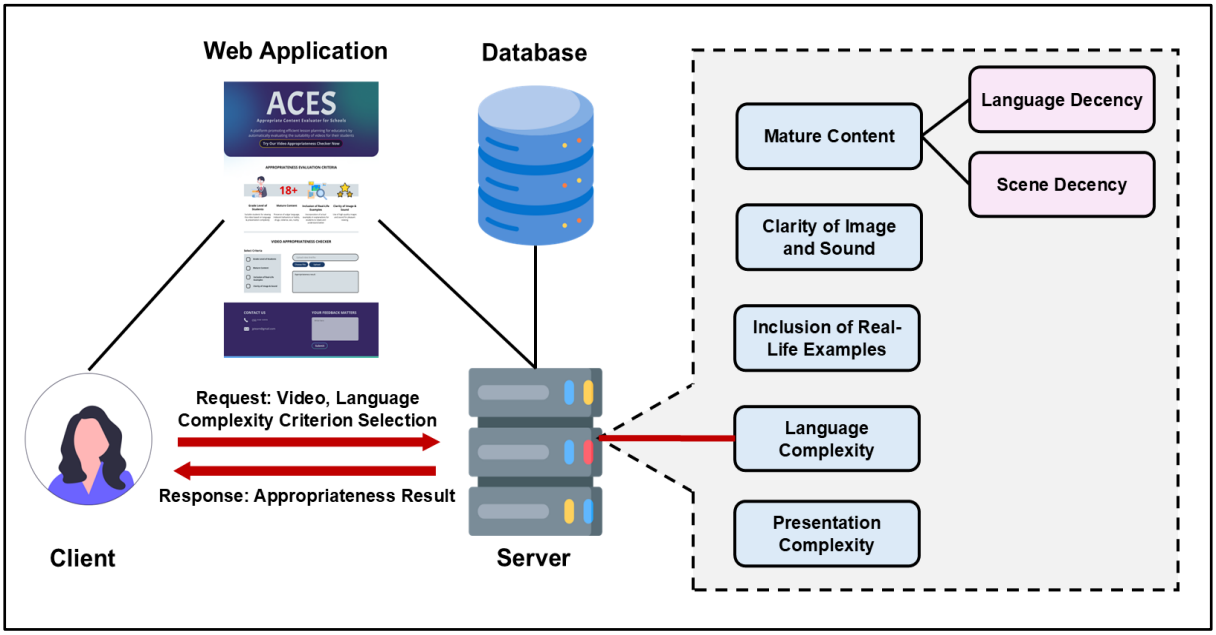

Figure 4-1: Overview of client-server software architecture

4.2 Web Application UX/UI Design Process

In web development, User Experience (UX) design focuses on enhancing the experience of the user interaction with the web application. On the other hand, User Interface (UI) design is the process of creating a visually pleasing interface that the user will use to interact. A combination of the two is essential in producing an intuitive and captivating user interface. Below outlines the key steps we followed in UX/UI design to make the user interface of our web application ACES [14].

1. **User flow diagramming** allows visualizing and planning user navigation through the application to ensure the user experience of using every feature is uncomplicated [15].

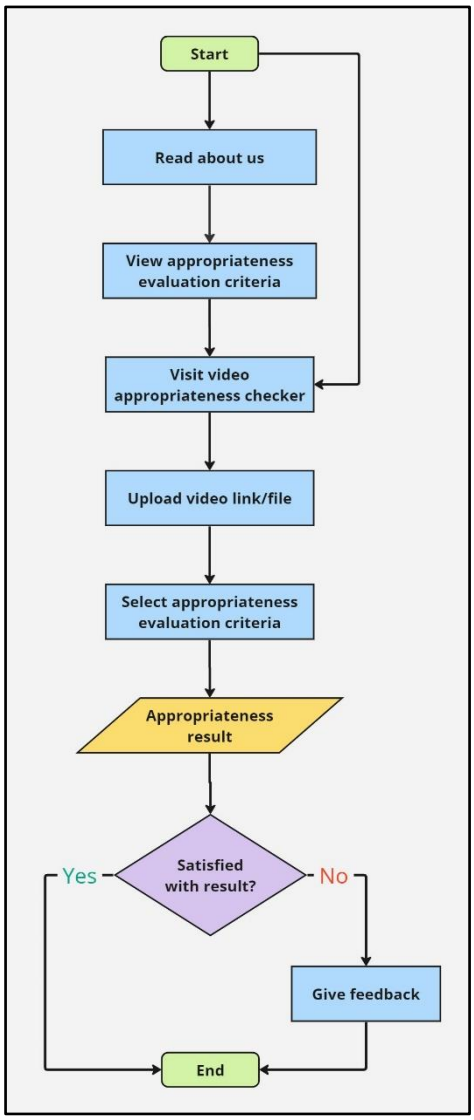

Figure 4-2: User flow diagram describing user navigation in the ACES web application

2. **Wireframing** is used for planning the layout of the application to ease the UI design process by knowing where each element should be placed beforehand [16].

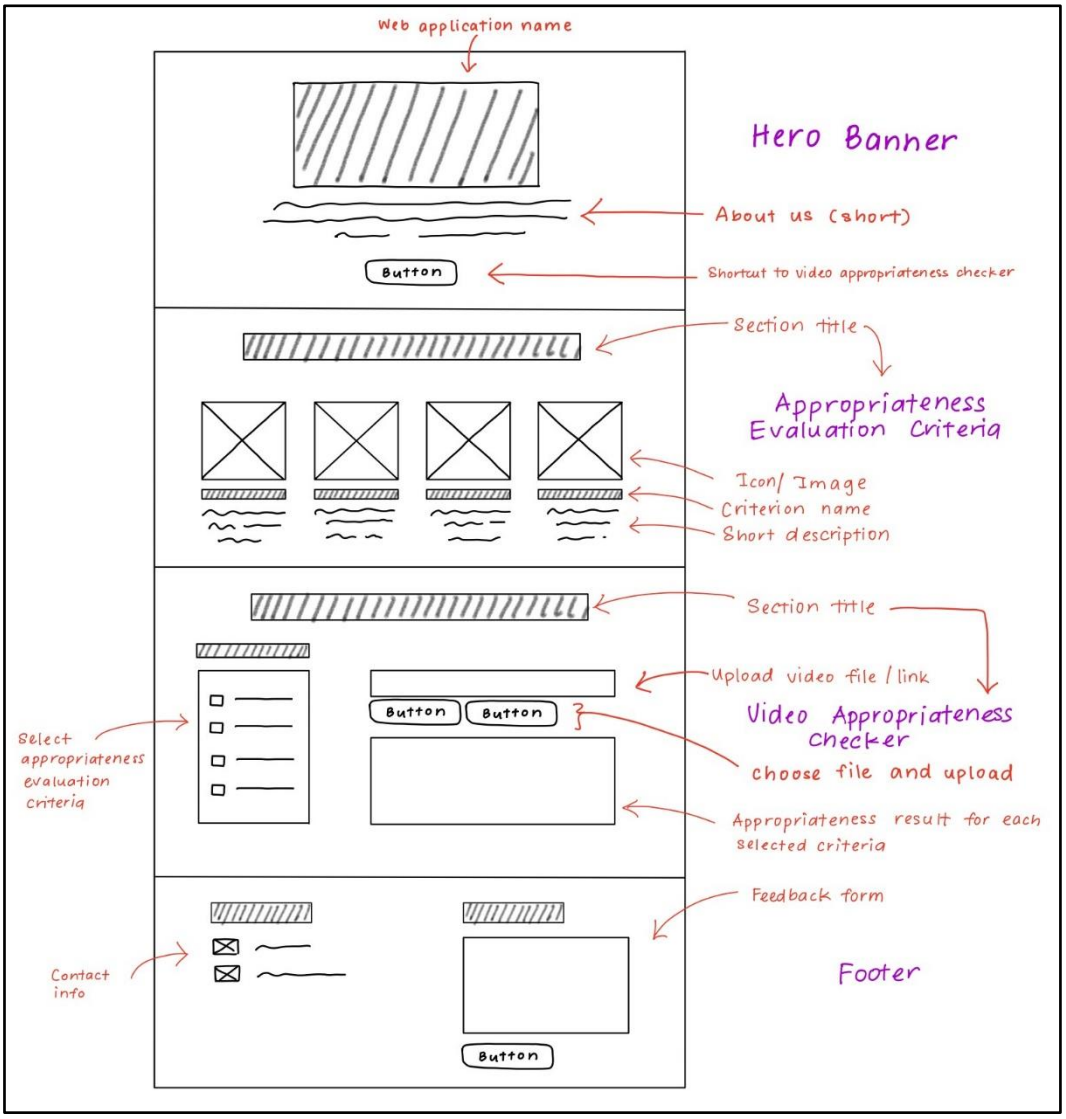

Figure 4-3: Wireframe illustrating the layout of the ACES web application

3. **User interface designing** for our web application was done on Figma which is a platform that supports drag-and-drop function to create prototypes easily.

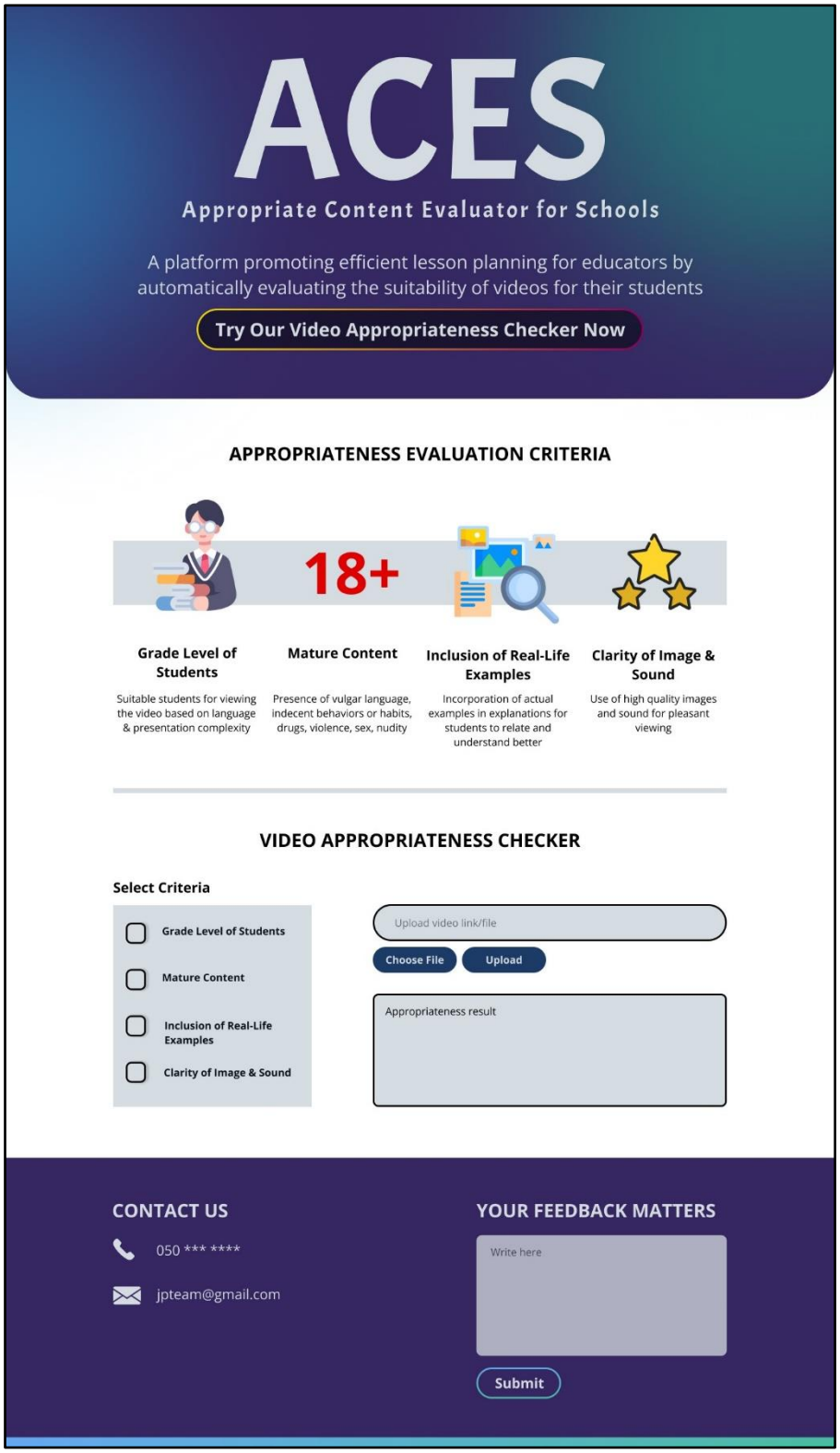

Figure 4-4: User interface of the ACES web application

CHAPTER 5: AI Model Implementation

5.1 Methodology

As the focus of this research is to help school teachers efficiently and effectively find videos best suited to the needs of students in various grade levels, a novel approach was considered, that is determining videos' appropriateness by examining the complexity of their content both visually and auditorily. In this project, we have implemented Presentation Complexity and Language Complexity criteria using Machine Learning and Deep Learning, respectively. Figure 5-1 and 5-2 summarizes the complete workflow of both predictors and the following sections provide a detailed description of the methodology.

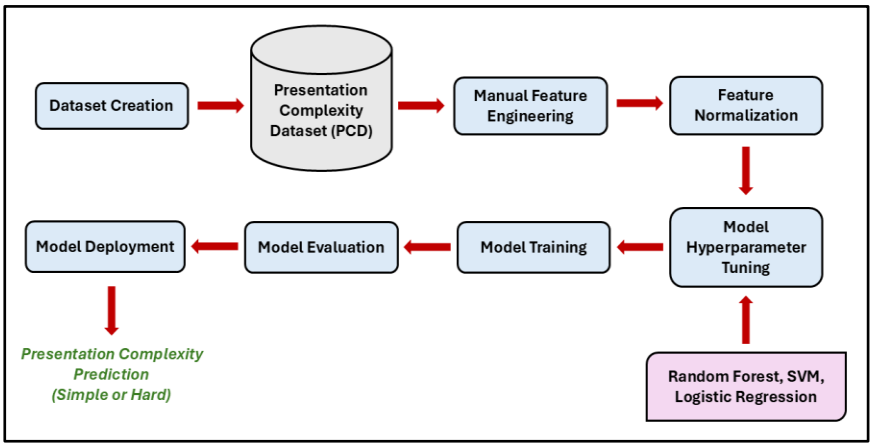

Figure 5-1: Workflow of the Presentation Complexity predictor

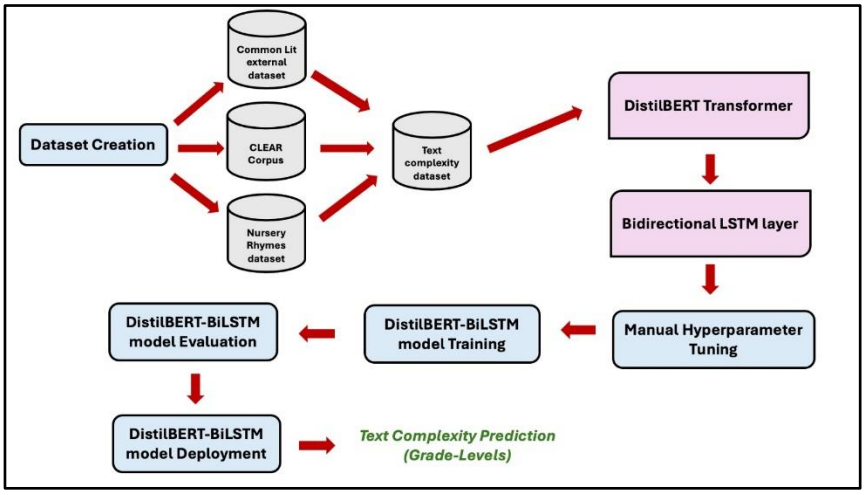

Figure 5-2: Workflow of the Language Complexity predictor

5.1.1 Presentation Complexity

A. Dataset Creation

Due to the unavailability of existing datasets related to video presentation complexity, we constructed our own dataset by collecting educational videos from YouTube in both simple and hard complexity levels. These complexity levels are the class labels that the model will aim to classify the videos into. The factors we accounted for distinguishing simple and hard videos were the amount of text, number of graphics and animations, and the organization of information. The process of data collection and annotation for creating the final dataset can be broken down into a series of steps. Firstly, we recorded the names and URLs of YouTube educational playlists on an Excel document and manually labelled them as simple or hard by roughly guessing the complexity levels of majority of the videos in those playlists. Then, the names and URLs of the videos in each playlist were extracted and weakly annotated using Python based on the playlist complexity level estimation. To save processing time, only the URLs of videos with a duration of under 11 minutes were obtained. In addition, we included a certain number of URLs from each playlist to have a balanced variety of presentation styles in the dataset for unbiased model performance.

For greater accuracy, the resulting videos were watched completely and re-annotated manually by three people (Figure 5-3). The weak labels were hidden, and the samples were shuffled to avoid any influence that could adversely affect the manual annotation phase. Videos with unmatching labels were either removed or re-labelled after a thorough discussion. A comparison between the manually annotated and weakly annotated labels showed insignificant difference, evidencing our concepts of simple and hard presentation styles did not vary at any point. This guarantees that the annotation process was consistent and fair.

Our final presentation complexity dataset contains 171 samples in total, and it is almost balanced as it has a class ratio of 1.2:1 with 94 videos in the ‘simple’ category and 77 videos in the ‘hard’ category. The ‘simple’ and ‘hard’ categorical class labels were replaced with numerical values of ‘0’ and ‘1’, respectively. Finally, the videos were downloaded in ‘mp4’ file format with the highest resolution using their URLs. The video name and URL extraction, and video downloading were automatically completed using the pytubefix Python library.

	C	D	E	F	G
1	**Video URL**	**Kavisna**	**Athifa**	**Sana**	**Same Labels?**
2	https://www.youtube.com/watch?v=bi2CLSuTmcs	hard	hard	hard	Yes
3	https://www.youtube.com/watch?v=jloEzVh31TE	simple	simple	simple	Yes
4	https://www.youtube.com/watch?v=xVf5kZA0HtQ	simple	simple	simple	Yes
5	https://www.youtube.com/watch?v=ghuuJNusnWE	hard	hard	hard	Yes
6	https://www.youtube.com/watch?v=QPQy7jUpmyA	simple	simple	simple	Yes
7	https://www.youtube.com/watch?v=GuXXSrj7r4U	simple	hard	hard	No
8	https://www.youtube.com/watch?v=Kxwdt0-wDyo	simple	simple	simple	Yes
9	https://www.youtube.com/watch?v=3c_dTtx7ob8	simple	hard	simple	No
10	https://www.youtube.com/watch?v=dANnzmxhIDE	hard	hard	hard	Yes

Figure 5-3: Manual re-annotation of the videos' presentation complexity by three people

B. Manual Feature Engineering

A total of 16 features were manually extracted from videos in the dataset using Python libraries, for the machine learning models to learn the unique patterns in the data to make predictions. Some of the features required extracting frames from the downloaded videos to be computed. The OpenCV Python library was used for frame extraction, with frames extracted at a rate of one frame per second (fps) including the first frame, instead of using the original frame rate. We followed this method as it was observed in videos with a frame rate of 30 fps for instance, that 30 frames within one second mostly contained redundant content, and significant visual changes were more likely to occur at every one second. Hence, using a rate of 1 fps for frame extraction reduced the number of extracted frames, while retaining the essential information by successfully capturing all the scenes with fewer repetitions. As a result, we were able to efficiently utilize the available computational resources.

In addition, these features were normalized after train-test splitting using Min-Max scaling technique to ensure all the features were on a similar scale for unbiased performance in distance-based algorithms, such as Support Vector Machine (SVM). Normalization was preferred over standardization due to the skewed distribution exhibited by all the features (Figure 5-4) [17]. The scaling transformation strategy was learned using only the training set, rather than the whole dataset, to prevent the test set values from impacting the training data, which could otherwise introduce bias into the model [18]. Then, the learned transformation was applied to both training and test sets' input features.

The manually extracted features and some of their formulas are as follows:

1. **Frame Rate (fps)** is the number of frames in one second, measured in fps.

2. **Scene Transition Rate (Per Minute)** is the number of scene transitions in one minute.

$$Scene\ Transition\ Rate = \frac{Total\ Number\ of\ Scene\ Transitions}{Video\ Length\ /\ 60} \tag{1}$$

where Total Number of Scene Transitions = (Number of Scenes – 1) if Number of Scenes is greater than zero, otherwise 0, and Video Length is given by:

$$Video\ Length = \frac{Total\ Number\ of\ Frames}{Frame\ Rate} \tag{2}$$

where Video Length is computed in seconds.

3. **Average Scene Duration (in Minutes)**

$$Average\ Scene\ Duration = \begin{cases} \frac{\sum_{i=1}^{S} Duration\ of\ Scene^{i}}{S}, & S > 0 \\ Video\ Length\ /\ 60, & S = 0 \end{cases} \tag{3}$$

where S is the total number of scenes and Video Length is computed using equation (2).
Note: Total number of scenes equal to zero implies that there is only one scene spanning the entire video, hence, the average scene duration is same as the video length.

4. **Average Motion Intensity** [19] is the average amount of movement detected in the entire video.

$$Average\ Motion\ Intensity = \frac{\sum_{i=1}^{F} f^{i}}{F} \tag{4}$$

where F is the total number of consecutive frame pairs and f^{i} is the average optical flow or motion vector magnitude of frame pair i, given by:

$$f^{i} = \frac{\sum_{j=1}^{P} p^{j}}{P} \tag{5}$$

where P is the total number of pixels in a frame and p^{j} is the optical flow or motion vector magnitude of pixel j between two consecutive frames.

5. **Average Texture Contrast** [20] is the average variation in the intensity between neighboring pixels in the frames of an entire video.

6. **Average Texture Homogeneity** [20] is the average uniformity of the texture in the entire video.

7. **Dominant Color Standard Deviation (R, G, B)** [21] is the standard deviation of the red (R), blue (B), and green (G) color channels of the dominant color, across the frames in a video, where each color channel is considered as a separate feature.

8. **Bit Rate (Mbps)** [22] is the amount of information in megabits, conveyed per second.

$$Bit\ Rate = \frac{(Video\ File\ Size\ \times 8)/\ Video\ Length}{10^6} \tag{6}$$

where Video File Size is in bytes and Video Length is computed using equation (2).

9. **Compression Ratio** [23] is a measure of how much the downloaded video file has been compressed compared to its original size on YouTube.

$$Compression\ Ratio = \frac{Compressed\ Video\ File\ Size}{Uncompressed\ Video\ File\ Size} \tag{7}$$

where Uncompressed Video File Size is given by:

$$Uncompressed\ Video\ File\ Size = W \times H \times B \times C \times Total\ Number\ of\ Frames \tag{8}$$

where W is the frame width, H is the frame height, B is the bit depth which is the number of bits used to represent each color channel, and C is the number of color channels.

10. **Average Peak Signal-to-Noise Ratio (PSNR) (dB)** [24] is the average ratio of the maximum signal power to the noise power between consecutive frame pairs in the entire video, measured in decibels (dB).

$$Average\ PSNR = \frac{\sum_{i=1}^{F} psnr^i}{F} \tag{9}$$

where F is the total number of consecutive frame pairs and $psnr^i$ is the PSNR of frame pair i, given by:

$$psnr^i = \begin{cases} 100, & if\ MSE < 10^{-10} \\ 20\ log_{10}(\frac{1}{\sqrt{MSE}}), & else \end{cases} \quad (10)$$

where MSE is the Mean Squared Error [25], and it is given by:

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(\frac{f(i,j)}{255} - \frac{g(i,j)}{255})^2 \quad (11)$$

where i is the row index in m rows of pixels; j is the column index in n columns of pixels; f is the first frame matrix; and g is the next consecutive frame matrix.

11. **Average Structural Similarity Index (SSIM)** [26] is the average similarity between consecutive frame pairs in the entire video.

$$Average\ SSIM = \frac{\sum_{i=1}^{F} s^i}{F} \quad (12)$$

where F is the total number of consecutive frame pairs and s^i is the similarity index of frame pair i.

12. **Average Color Standard Deviation (R, G, B)** [27] is the standard deviation of the red (R), blue (B), and green (G) color channels of the average color in frames of a video, where each color channel is considered as a separate feature.

An overview of the dataset including manually extracted features and their computation tools, and the composition of the dataset are presented on Table 5-1 and 5-2, respectively.

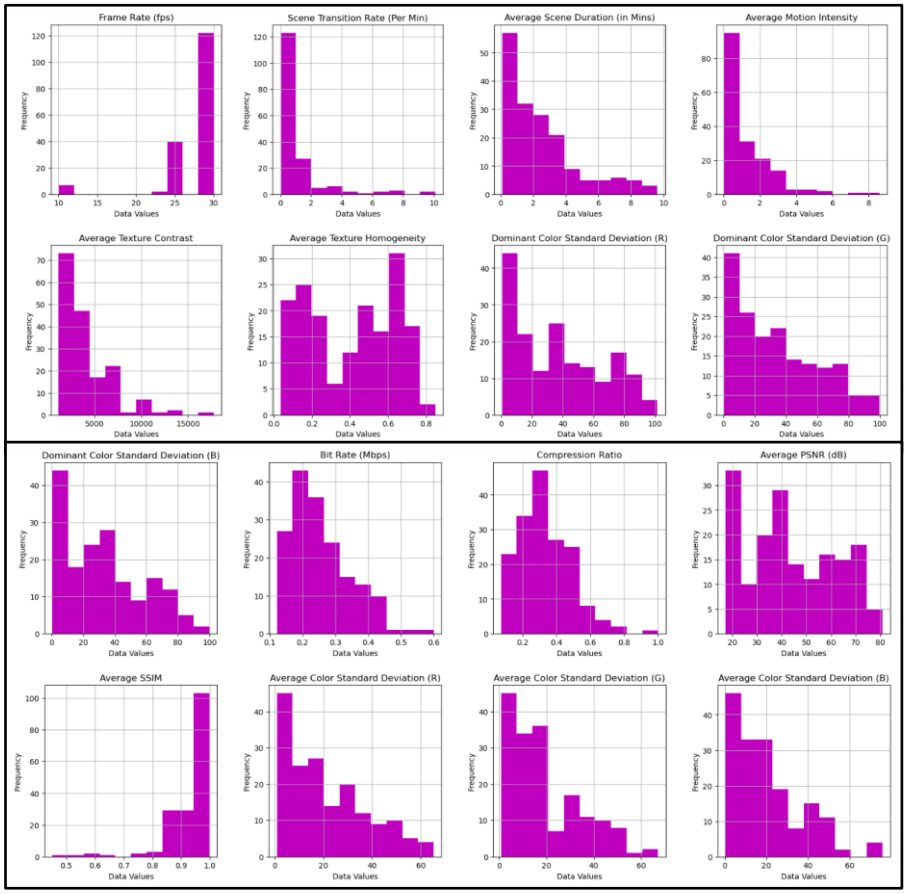

Figure 5-4: Distribution of the manually extracted features

Table 5-1: Overview of presentation complexity dataset including manually extracted features and their computation tools

Features	Computation Tools
Video Name	N/A
Video URL	N/A
Complexity Level	N/A
Manual Features	
Frame Rate (fps)	OpenCV library
Scene Transition Rate (Per Minute)	Adaptive Detector algorithm in PySceneDetect library [28] was used to detect scenes based on changes in color and intensity between frames, with minimal risk of detecting false scenes, and OpenCV library was used to measure the video length
Average Scene Duration (in Minutes)	
Average Motion Intensity	Gunnar Farneback algorithm in OpenCV library was used to compute the dense optical flow, which is the motion vector of all pixels between two consecutive frames, and NumPy library was used to find the average

Average Texture Contrast	Gray Level Co-occurrence Matrix (GLCM) method in scikit-image library [29] was used to compute the texture features, and NumPy library was used to find the average
Average Texture Homogeneity	
Dominant Color Standard Deviation in Red (R), Green (G), and Blue (B) color channels	Fast colorthief library was used to determine the dominant color, and NumPy library was used to calculate the standard deviation
Bit Rate (Mbps)	OS and OpenCV libraries were used to get the video file size and length, respectively
Compression Ratio	OpenCV and OS libraries were used to get the uncompressed and compressed video file sizes, respectively
Average Peak Signal-to-Noise Ratio (PSNR) (dB)	NumPy and Math libraries
Average Structural Similarity Index (SSIM)	Structural Similarity method in scikit-image library was used to compute the similarity index based on the differences in brightness, contrast, and structure between frames, and NumPy library was used to find the average
Average Color Standard Deviation in Red (R), Green (G), and Blue (B) color channels	NumPy library

Table 5-2: Composition of the presentation complexity dataset

Total No. of Samples	171
Total No. of Features	19
Total No. of Manually Extracted Features	16
No. of Samples in 'simple' Class	94
No. of Samples in 'hard' Class	77

C. Machine Learning Classification

The supervised machine learning algorithms employed for the binary classification task of predicting video presentation complexity level were Random Forest, Support Vector Machine (SVM), and Logistic Regression. The manually engineered features and the complexity level serve as the input and output variables, respectively, which will be used for training and testing the models. These variables were split into training and test sets using a 70/30 ratio in a stratified manner to maintain the same class ratio in both sets.

GridSearchCV-based hyperparameter tuning was performed on all models using the unscaled training set to identify the parameters that provide the best performance. The GridSearchCV was implemented using stratified 5-fold cross-validation and f1-score as the scoring method to evaluate the performance of each possible combinations of parameters on different subsets of training data. With stratified cross-validation, the class ratio remains the same in the resulting training and validation sets after the split of the training data during cross-validation.

Since cross-validation further divides the training data into training and validation sets, the data must not be normalized beforehand, as this could leak validation set's information into the training set. Therefore, the unscaled training set was used as input to the GridSearchCV method. Normalization was applied after the data splitting stage of cross-validation using a pipeline, which systematically specifies the data preprocessing step [18].

After hyperparameter tuning, the models were trained with their best parameters (Table 5-3) using the normalized training set, followed by evaluation of their performance on both normalized training and test sets to check for overfitting and underfitting issues. Also, the importance of each input feature during training and testing was determined by using permutation importance score to understand the models' predictions. Permutation importance measures the model's performance decrease when a feature's values are randomly permuted or shuffled. We used accuracy to measure the model's performance when computing permutation importance score for each feature, where a higher score correlates to a greater decrease in accuracy, demonstrating that the model's performance is significantly affected by shuffling. In this case, the feature is considered important with the permutation score reflecting the degree of its importance [30]. On the other hand, an unimportant feature will have a zero or negative permutation importance score, indicating either a no change in accuracy or a higher accuracy after shuffling [31]. A model can also be concluded as overfitted when there is large difference between the permutation importance scores of training and test sets [30]. The models' performance and feature importance results are discussed in detail in the Results and Discussion section.

Table 5-3: Best parameters for Random Forest, SVM, and Logistic Regression

Random Forest	
n_estimators	200
max_depth	None
min_samples_split	2
min_samples_leaf	1
max_features	None
max_leaf_nodes	None
SVM	
C	1.0
kernel	‘poly’
degree	2
gamma	1
Logistic Regression	
penalty	‘l2’
solver	‘liblinear’
C	10
max_iter	100

5.1.2 Language Complexity

A. Dataset Description

In this work, we aim to achieve grade-level classification into classes of kindergarten, grades 1-3, grades 4-6, grades 7-9, and grades 10-12. We achieved this by integrating two primary datasets.

The first dataset, the CommonLit External Dataset 2021 [32], specifically leverages the "children_stories.csv" component, curated from highly rated children's literature. From the CommonLit External Dataset, samples with age groups of maximum 5 were taken as kindergarten class.

Second, the CommonLit Ease of Readability (CLEAR) corpus [33], offers unique readability scores for approximately 5,000 samples. The corpus encompasses a diverse range of texts spanning 250 years and two genres, providing valuable insights into text complexity based on teacher ratings. To categorize the samples into different grade levels the Automated

Readability Index (ARI) score, a well-established metric for assessing text complexity, was employed to assign the grade-levels. ARI provides an estimated measure of reading difficulty based on word and sentence lengths, with each score corresponding to a specific grade level (Table 5-4). ARI is calculated using a mathematical formula that considers the average number of characters per word and the average number of words per sentence in each text, as shown in equation (13).

$$Automated\ Readability\ Index = 4.7\left(\frac{characters}{words}\right) + 0.5\left(\frac{words}{sentence}\right) - 21.43 \quad (13)$$

Table 5-4: ARI score table for Kindergarten to grade level 12.

ARI Score	Grade Level
3 and below	Grade 1-3
4-6	Grade 4-6
7-9	Grade 7-9
10-12	Grade 10-12

After the integration of the mentioned dataset, we have noticed that the class of kindergarten is significantly less. So, we have added third dataset, English Nursery Rhymes dataset [34]. The dataset was created with intension for it to be used as a learning tool for teaching natural language processing techniques, containing 308 samples. The final datasets sample count can be seen in Table 5-5.

Table 5-5: Original imbalanced dataset sample count.

Grade Category	Number of Text Samples
Grade 10-12	2,382
Grade 7-9	1,108
Grade 4-6	738
Grade 1-3	496
Kindergarten	476
Total	5,200

B. Data Preprocessing

To prepare the combined annotated dataset for subsequent analysis, we applied a series of text preprocessing techniques. Text preprocessing involves a range of methods to clean, transform, and standardize raw textual data, making it suitable for natural language processing

(NLP) or machine learning (ML) tasks [35]. The primary goal of text preprocessing is to enhance the quality and usability of the text data for analyzing text complexity.

The text preprocessing process employed in this study included lowercasing, punctuation removal, special character elimination, stop word removal, URL and HTML tag removal, stemming and lemmatization, tokenization, and text normalization [35]. These techniques contribute to feature engineering, creating a structured representation of the text data that facilitates subsequent analysis.

While traditional machine learning models often require extensive pre-processing, such as feature engineering, the DistilBERT model leverages its built-in pre-processing capabilities. This allows the model to directly process raw text data without the need for manual feature extraction. By analyzing the context of the text, DistilBERT can capture subtle nuances and complexities, making it a powerful tool for various NLP tasks. So, we have decided to use the model without integrating manual feature engineering.

Tokenization, the process of breaking down text into smaller units called tokens, is a fundamental step in natural language processing (NLP). These tokens, which can be words, subwords, or characters, are essential for language models like DistilBERT to understand and process text data effectively. In our work, we utilized a pre-trained DistilBERT tokenizer to efficiently tokenize our text data. This tokenizer, specifically designed for the DistilBERT architecture, breaks down text into meaningful tokens, enabling the model to capture the semantic and syntactic structure of the text.

C. Manual Feature Engineering

We focused on a selection of linguistic features relevant to text complexity and readability, based on established frameworks, as referenced in Table 5 of Li et al. (2023) [36]. Specifically, we extracted the 14 features (Table 5-6) using Python packages NLTK and text descriptives. These features were grouped by grade category, enabling the model to identify patterns and trends in readability metrics that correspond to specific educational stages. We normalized all features through manual feature engineering using Min-Max scaling.

Table 5-6: Manual linguistic features

Name of the Feature
avg_words_per_sentence
avg_syllables_per_word
avg_letters_per_word
avg_conjunctions_per_text
percentage_pronouns
avg_long_form_punctuation_count
np_percentage (noun)
pnp_percentage(proper noun)
avg_noun_phrase_length
avg_noun_phrases_per_sentence
avg_prepositional_phrase_length
avg_prepositional_phrases_per_sentence
avg_verb_phrase_length
avg_verb_phrases_per_sentence

D. Model Architecture

To analyze the language complexity of the videos, we first extracted audio tracks using moviepy library and subsequently converted them into text using the Whisper model developed by OpenAI. The resulting text was then processed by DistilBERT to assess its complexity (Figure 5-5).

Whisper models are open-source and have been pre-trained by OpenAI for automatic speech recognition and translation. Whisper large-v3 is the latest version with an accuracy of 10-20% greater than the Whisper large-v2. The audios extracted in step 3 were converted to texts using the Whisper large-v3 model from Hugging Face. Prior to this, the 38-sample rate of the audios were changed to adhere to the requirements of the Whisper model. The implementation of this model was done using the PyTorch framework [37].

To analyze the linguistic complexity of the extracted text, we implemented two models: one incorporating additional features and another serving as a baseline. By comparing their performance, we aimed to determine the optimal approach for assessing language complexity.

The first model, without additional features, will serve as a baseline, allowing us to evaluate the impact of the supplementary features on the model's performance. We opted for

the DistilBERT framework, specifically the distilbert-base-cased model, for text classification across five grade categories. It efficiently balances performance and computational resources, making it particularly suitable for handling our dataset. DistilBERT, as a distilled version of BERT, retains much of the original model's language understanding capabilities while being smaller and faster [38]. The model utilizes the DistilBert Model, which comprises an embedding layer that includes word and position embeddings, followed by a series of six transformer blocks. Each transformer block features multi-head self-attention mechanisms, feed-forward networks (FFN), and normalization layers, enabling the model to capture complex relationships within the input data. The embeddings generate rich contextual representations of input tokens, while the transformer layers process these representations through attention mechanisms and non-linear transformations. The output of the last transformer layer is then fed into a fully connected linear layer, mapping the 768-dimensional embeddings to the desired number of classes—in our case, five. This architecture effectively leverages the strengths of DistilBERT's pre-trained capabilities, allowing for efficient learning and accurate predictions across multiple classes [39].

For our second model, we explored the integration of manually engineered features into the DistilBERT framework. However, we observed a potential conflict between the model's contextual understanding of text with punctuations and the pre-processed nature of the manually engineered features. This discrepancy could potentially lead to confusion within the model. As a result, we opted to utilize the DistilBERT model without incorporating the manually engineered features.

To enhance language complexity analysis, we incorporated an LSTM-based architecture using TensorFlow. The model processes DistilBERT embeddings through a Bidirectional LSTM with 128 units to capture both past and future context. The LSTM output is then passed through a GlobalMaxPool1D layer to focus on the most relevant features. Next, a Dense layer with 64 units and ReLU activation is followed by a Dropout layer (0.2 rate) to prevent overfitting. The final Dense layer with a number of units equal to the number of classes and softmax activation outputs probabilities for each class. We froze the first few layers of the model to retain DistilBERT's pre-trained features, allowing the model to focus on the LSTM and fully connected layers during training.

To implement the model, we employed a stratified random split, allocating 70% of the data to the training set, 15% to the validation set, and the remaining 15% to the testing set. This approach ensured a balanced representation of classes across all sets. The model was trained for a maximum of 16 epochs, but early stopping was implemented to prevent overfitting.

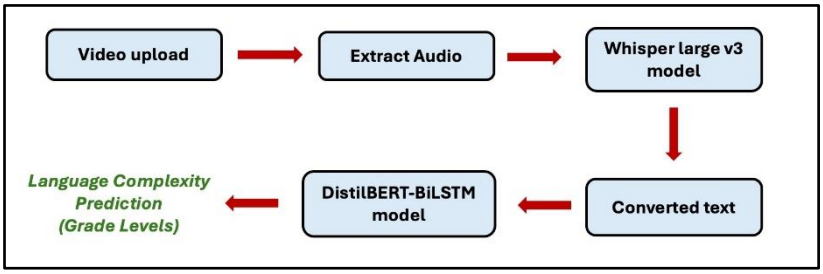

Figure 5-5: Language complexity workflow

E. Hyperparameter Tuning

Hyperparameter tuning plays a pivotal role in achieving optimal model performance. Two key techniques, Early Stopping and Learning Rate Reduction, are employed to enhance the training process and prevent overfitting.

Early Stopping is a strategy that halts the training process when the validation loss fails to improve for a specified number of epochs. This prevents the model from overfitting to the training data, leading to better generalization on unseen data. By monitoring the validation loss, the algorithm can identify the point where further training yields diminishing returns, the point where additional training of the model no longer significantly improves its performance.

Learning Rate Reduction, also known as Learning Rate Decay, is a technique that gradually decreases the learning rate during training. This helps the model to converge more smoothly and avoid getting stuck in local minima. In our model, we employed a learning rate of 1e-4. By reducing the learning rate, the model takes smaller steps, allowing it to fine-tune its parameters more precisely.

The selection of the optimizer and batch size significantly influences the training process. In this particular instance, the Adam optimizer was employed. This optimizer incorporates a weight decay of 1e-4 to enhance regularization and mitigate the risk of overfitting. Additionally, a batch size of 16 was utilized, which provides a balance between computational efficiency and gradient estimation accuracy.

When used in conjunction, Early Stopping and Learning Rate Reduction can significantly improve the training process. Early Stopping can prevent unnecessary training time, while Learning Rate Reduction can enhance the model's convergence. By carefully selecting the optimizer, batch size, and employing techniques like Early Stopping and Learning Rate Reduction, we optimized the training process, improve model performance, and reduce computational costs.

5.1.3 Evaluation Metrics

The metrics used to evaluate the performance of binary classifiers in Presentation Complexity and the multi-class classifier in Language Complexity are accuracy, precision, recall (sensitivity), f1-score, and ROC AUC, which is the area under Receiver Operating Characteristic (ROC) curve. Additionally, the proportion of correct and incorrect classifications were checked using a confusion matrix. Since there was class imbalance in the datasets, normalized confusion matrix and macro-averaged precision, recall, and f1-score were used. A normalized confusion matrix shows the classification result with respect to the number of samples in a specific class. Macro-averaging measures how well the model performs across all classes with equal weights or importance given to each class. Each class's metrics are computed separately and later aggregated to find the unweighted average. Accuracy is the proportion of correct predictions over the total predictions made by the model, and it is calculated as:

$$Accuracy = \frac{1}{C}\sum_{i=1}^{C}\frac{TP^i + TN^i}{TP^i + FP^i + TN^i + FN^i} \tag{14}$$

where C is the total number of classes; TP is the number of true positive samples, TN is the number of true negative samples; FP is the number of false positive samples; and FN is the number of false negative samples. Precision is the proportion of correct positive predictions over the total positive predictions made by the model, and it is calculated as:

$$Precision = \frac{1}{C}\sum_{i=1}^{C}\frac{TP^i}{TP^i + FP^i} \tag{15}$$

Recall or sensitivity is the proportion of correct positive predictions over the total number of actual positive samples, providing information on how well the model predicts positive samples. It is calculated as:

$$Recall = \frac{1}{C}\sum_{i=1}^{C}\frac{TP^i}{TP^i + FN^i} \tag{16}$$

F1-score is the harmonic mean of precision and recall, quantifying how well-balanced the model is at predicting both positive and negative classes, and it is calculated as:

$$F1Score = 2 \times (\frac{Precision \times Recall}{Precision + Recall}) \tag{17}$$

ROC AUC is a measure of a model's ability to distinguish between positive and negative samples. However, this metric is natively designed for binary classification tasks. To adapt it to the multi-class classification problem in Language Complexity, the One-vs-Rest (OvR) approach is used, where ROC AUC for each class is computed by treating that class as positive and the rest of the classes as negative. The results across all classes are then averaged to obtain the average ROC AUC. Additionally, the training and validation losses in the Language Complexity predictor were calculated using the multi-class cross-entropy loss method. This loss function defines the difference between the actual and predicted outcomes, or in other words, the error. It is calculated as:

$$Loss = -\sum_{i=1}^{C} y^i \log(p^i) \tag{18}$$

where y^i is the ith actual label in the actual outcome vector y and p^i is the ith predicted probability in the predicted outcome vector p.

5.2 Results and Discussion

5.2.1 Presentation Complexity

The trained and fine-tuned models were evaluated on both training and test sets to ensure that the models were neither overfitting nor underfitting, and their results are displayed in Table 5-7. All models achieved high performance, with only small differences between them. However, the results of SVM on the test set is 1.0 in all evaluation metrics, which is slightly greater when compared to the training set. Although a model is generally said to be overfitted when the performance on the training set is significantly greater than the test set, our results of SVM also show a form of overfitting. To elaborate, due to the small test set size with only

around 30 samples in each class, there may have been samples with almost identical patterns to the training set samples. Hence, the SVM model is able to predict the complexity level of each sample accurately as it may have perfectly memorized the training data or, in other words, it overfitted to the training data. On the other hand, Random Forest and Logistic Regression do not display any sign of overfitting. Random Forest is performing better than Logistic Regression on both training and test sets (Table 5-7), with a slightly higher percentage of correct classification in 'hard' class for test set than that of Logistic Regression (Figure 5-6). Figure 5-7 and 5-8 illustrate the performance of each model on the test set, and their ROC curves for the visualization of ROC AUC metric, respectively.

As for the feature importance (Figure 5-9), overfitting in SVM have also been demonstrated by the large difference in the permutation importance score of each feature between training and test sets. Between Random Forest and Logistic Regression, Logistic Regression utilizes more features than Random Forest to make predictions. Both have the features Average PSNR (dB), Frame Rate (fps), Average Texture Homogeneity, and Average Color Standard Deviation (R) ranked with high importance.

All in all, Random Forest is concluded to be the best-performing model due to several reasons. It does not include overfitting like SVM. A high score for all evaluation metrics on both training and test sets were obtained. On top of that, it depends on only few features when predicting unseen samples' class, unlike Logistic Regression. Hence, feature selection for Random Forest may potentially decrease the model complexity, while maintaining good performance.

Table 5-7: Results of evaluation metrics for Random Forest, SVM, Logistic Regression

Evaluation Metrics	Random Forest		SVM		Logistic Regression	
	Training Set	Test Set	Training Set	Test Set	Training Set	Test Set
Accuracy	1.0	0.981	0.966	1.0	0.966	0.942
Macro Average Precision	1.0	0.979	0.966	1.0	0.966	0.944
Macro Average Recall	1.0	0.983	0.966	1.0	0.966	0.939
Macro Average F1-Score	1.0	0.981	0.966	1.0	0.966	0.941
ROC AUC	1.0	0.999	0.998	1.0	0.993	0.988

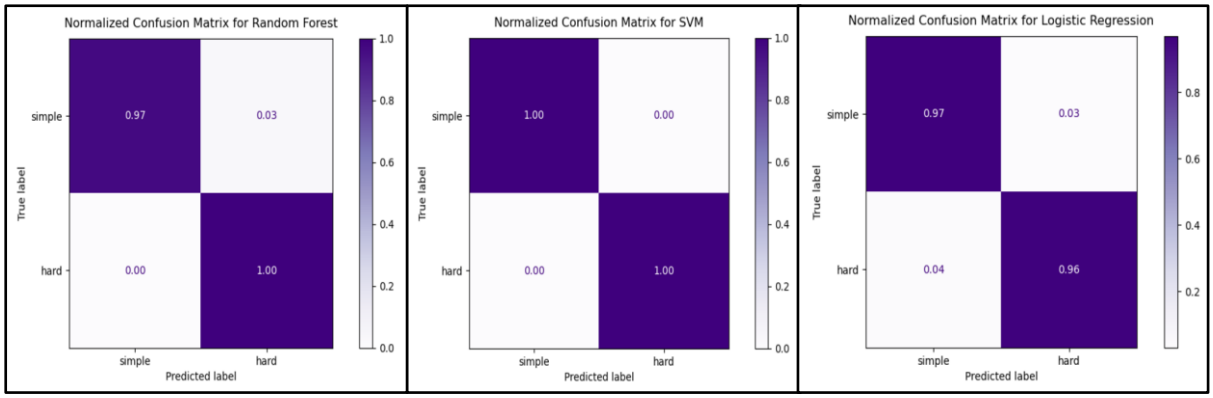

Figure 5-6: Normalized confusion matrix for Random Forest, SVM, and Logistic Regression on the test set

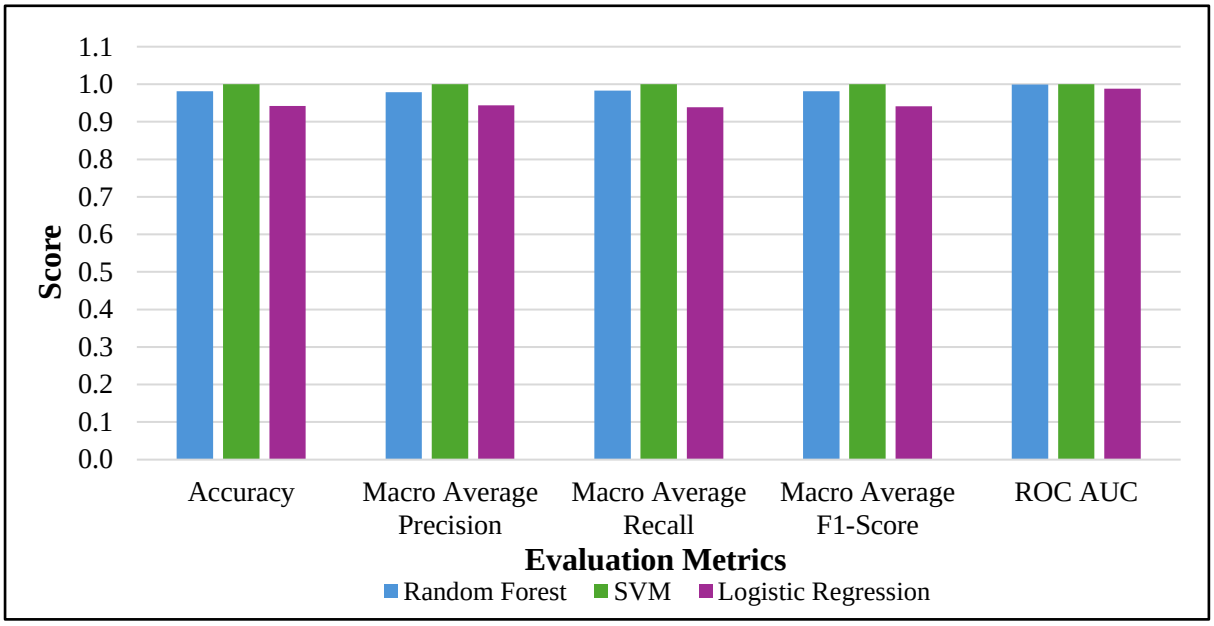

Figure 5-7: Performance comparison of Random Forest, SVM, and Logistic Regression on the test set

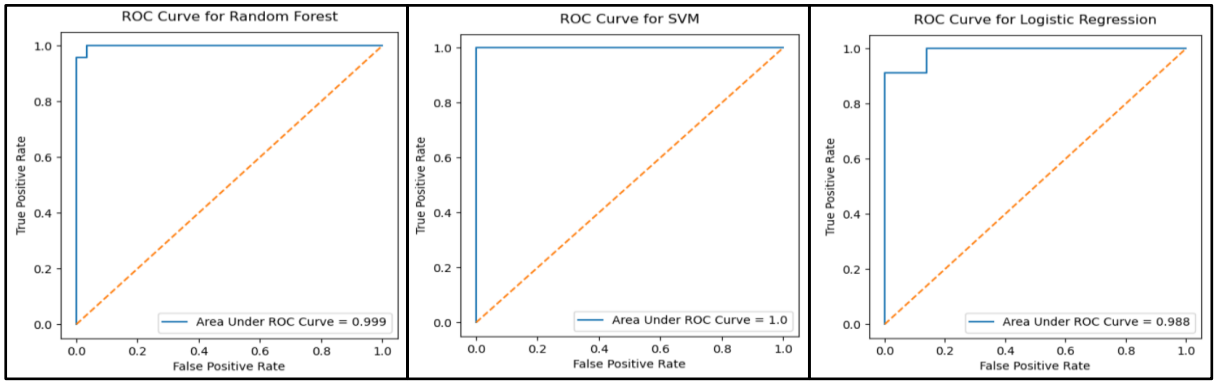

Figure 5-8: ROC curve for Random Forest, SVM, and Logistic Regression on the test set

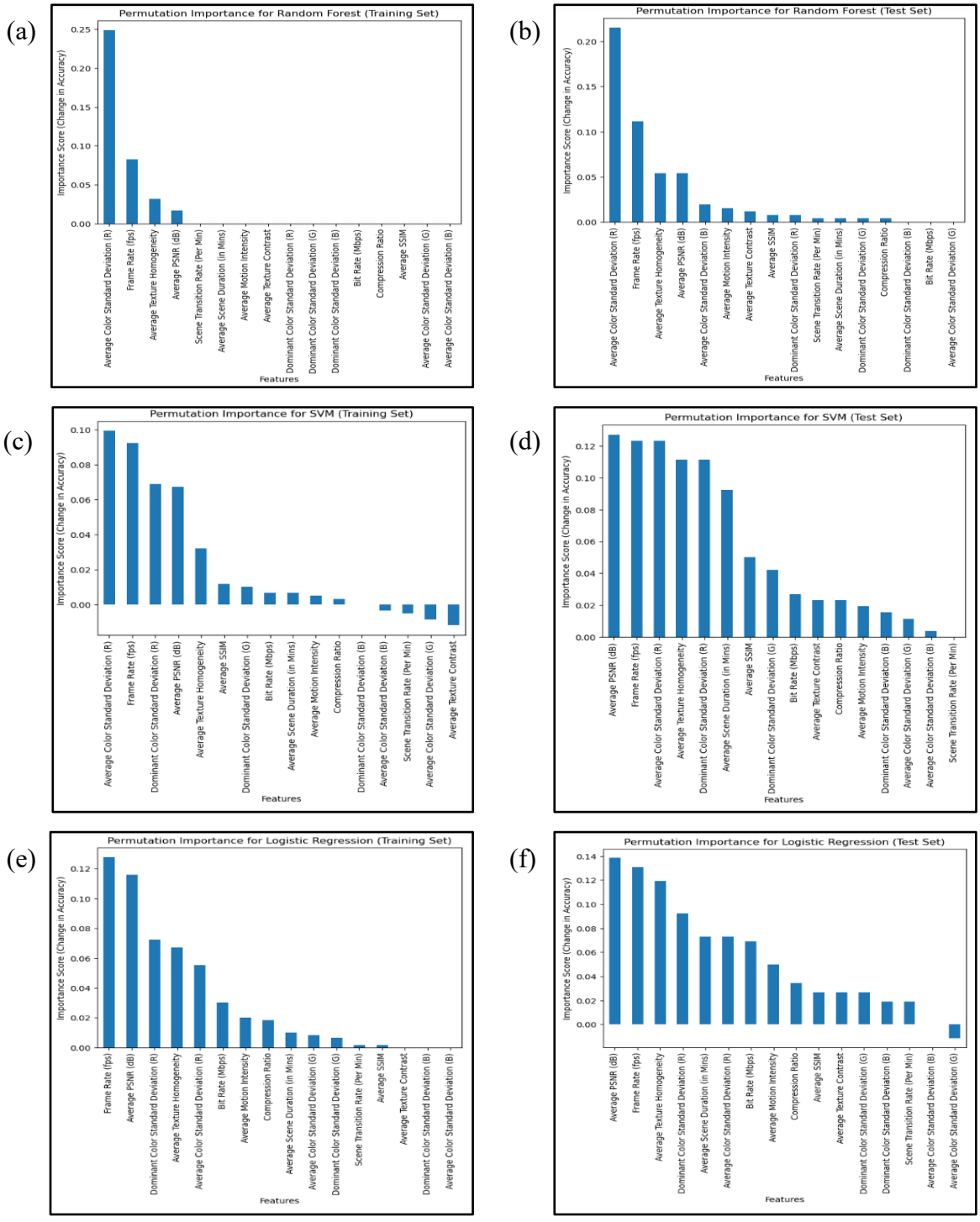

Figure 5-9: Feature importance in Random Forest (a) on the training set, (b) on the test set, SVM (c) on the training set, (d) on the test set, and Logistic Regression (e) on the training set, (f) on the test set

5.2.2 Language Complexity

The model's performance was evaluated on both training and validation datasets. The model with the highest validation F1-score, achieved at epoch 12, was selected as the final model. This final model demonstrated a high training F1-score of 97% and a validation F1-score of 83%. To visualize the training process, the training and validation loss, along with the F1 score curves, can be seen in Figure 5-10.

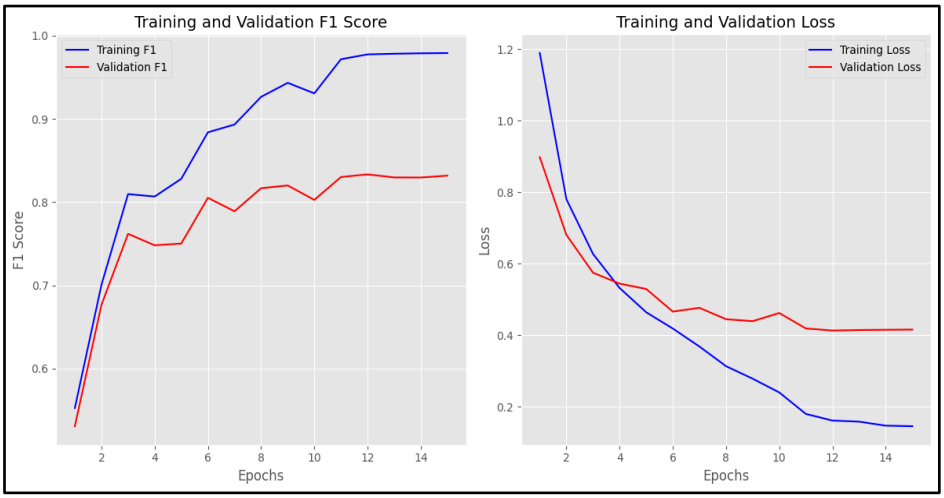

Figure 5-10: Training and validation loss, and F1 score curves for data without augmentation.

To further enhance the model's performance, we implemented data augmentation by randomly swapping 10% of non-stop words within each sample. This technique aimed to increase the sample size for each class to 1,000, potentially improving the model's generalization ability. However, despite these efforts, the F1 score remained unchanged, as shown in Figure 5-11.

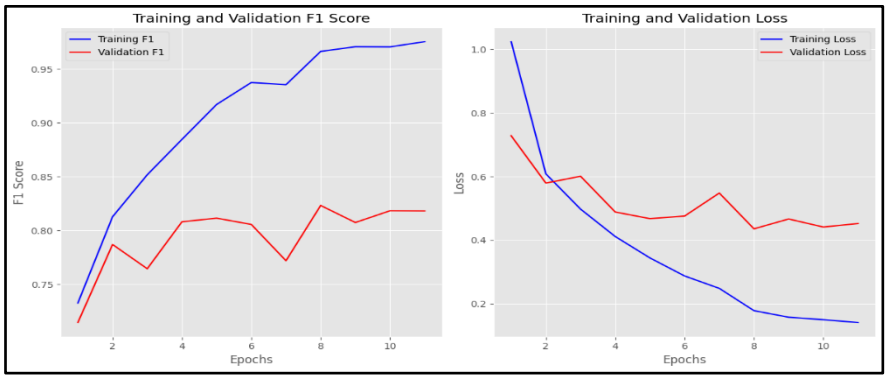

Figure 5-11: Training and validation loss, and F1 score curves for data with augmentation.

The model was evaluated on the unseen test set, achieving an accuracy of 82% without data augmentation and 81% with data augmentation. The performance of the model on unseen data is visualized through the normalized confusion matrices presented in Figures 5-12 and 5-13, respectively. These figures provide insights into the model's ability to correctly classify samples and identify potential areas for improvement.

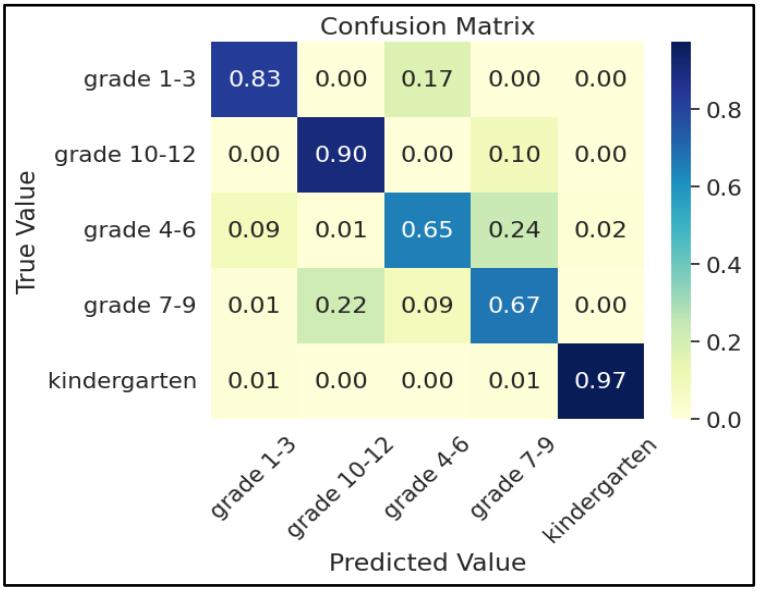

Figure 5-12: Normalized Confusion matrix for data without augmentation.

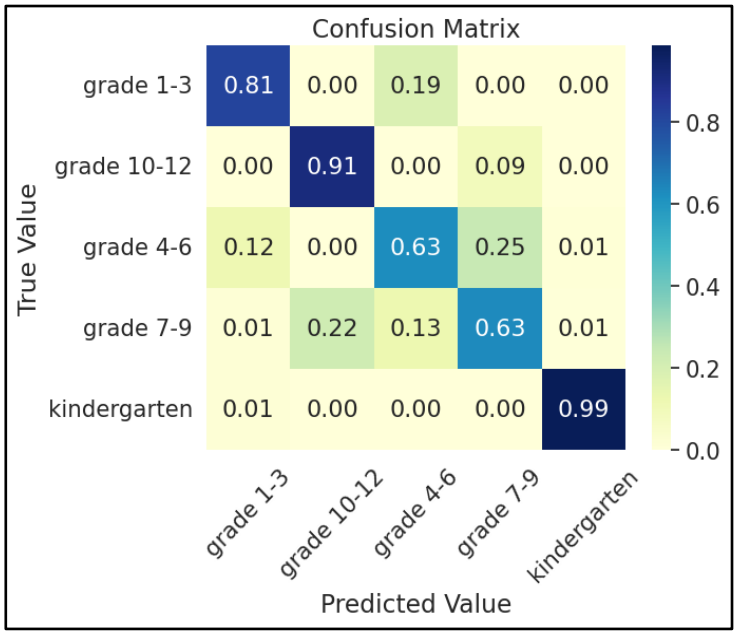

Figure 5-13: Normalized confusion matrix for data with augmentation.

5.3 Additional Model Testing and Improvement

5.3.1 Presentation Complexity

Since the original dataset's test set was very small, with only approximately 20 samples in each class, we conducted further testing on the selected trained and fine-tuned Random Forest model using newly collected YouTube videos. Due to changes in YouTube's processing script, as presumed by some developers in online forums, the pytubefix library which was initially used to automatically download the videos in the dataset, was no longer functioning. Other video downloading libraries also produced errors, hence we resorted to manually downloading the test videos using an online website [40].

The test dataset, consisting of 30 videos in total and 15 videos in each class, was manually annotated and verified by three people to minimize bias. Then, video frames were extracted, followed by the manual engineering of 16 features. These features were normalized using the Min-Max scaler that was fitted on the training data, before passing as inputs into the model for predictions. Accuracy and the normalized confusion matrix were used to evaluate the model's performance on the test data, as these metrics are known to be effective for balanced datasets. Despite achieving a high accuracy of 0.733, the confusion matrix (Figure 5-14) illustrates that this increase was largely driven by the model's ability to correctly classify 93% of simple videos. However, the majority of the hard videos were misclassified as simple, indicating the need for further model improvement.

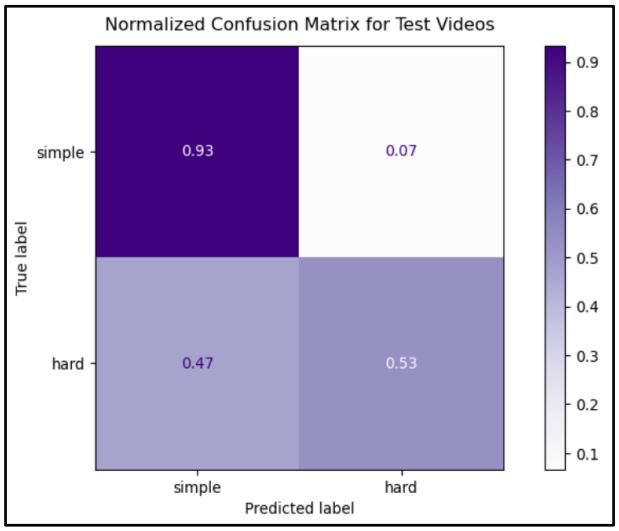

Figure 5-14: Normalized confusion matrix for test dataset.

The primary factor influencing the performance of AI models is the data on which they were trained. For this reason, we believe that the insufficient training data may be the cause of the poor performance on the new test data. To improve, we started by focusing on enlarging the original dataset. A total of 90 new videos from YouTube were collected, manually annotated and verified by three people. After performing manual feature engineering on these videos, they were merged with the original dataset and the new test dataset, both of which had already undergone feature engineering. The resulting dataset (Table 5-8) was balanced and comprised 291 samples, with 146 in the hard class and 145 in the simple class. The dataset was split into training and test sets using 70:30 ratio. The number of training samples significantly increased, from 65 to 101 in the simple class, and 54 to 102 in the hard class.

A new Min-Max scaler was fitted on the training data, which was then used for normalizing the input features in the training and test sets. Following this, hyperparameter tuning using GridSearchCV identified the best parameters for Random Forest, SVM, and Logistic Regression (Table 5-9). The models were trained using the best parameters and the normalized training set.

The results of the evaluation metrics (Table 5-10) and feature importance (Figure 5-15) show acceptable difference between the training and test sets, indicating that there was no overfitting or underfitting. The visualization of evaluation metrics results on test set (Figure 5-16) highlights Logistic Regression as the worst-performing model, mainly due to its poor classification of 45% of simple videos (Figure 5-17). On the other hand, both Random Forest and SVM performed almost equally well (Figure 5-16) and demonstrated good ability in accurately classifying simple and hard videos (Figure 5-17).

To select the best model between Random Forest and SVM, we considered the area under ROC curve, which quantifies the model's ability to distinguish between positive and negative classes, along with the feature importance results. Random Forest has an area under the ROC curve of 0.965, which is higher than the 0.956 for SVM (Figure 5-18). According to the feature importance graphs (Figure 5-15), the Random Forest relied heavily on a smaller set of features compared to SVM. Therefore, feature selection is more likely to yield positive results for Random Forest than for SVM. With feature selection, the essential features can be identified, potentially reducing the complexity of the model and the processing time. Since the results support Random Forest, it is concluded to be the best-performing model.

Furthermore, increasing the dataset size produced a larger test set after splitting, with the number of samples rising from 29 to 44 in the simple class, and from 23 to 44 in the hard class. Due to this, as well as the time constraint, we did not perform any further testing on new videos. Additionally, the model's performance could have also been affected by the accuracy of the annotations. An expert in the field of educational video creation would have better annotated the datasets. However, we were not able to find one, and this is one of the limitations of this project.

Table 5-8: Composition of the new presentation complexity dataset.

Total No. of Samples	291
Total No. of Features	19
Total No. of Manually Extracted Features	16
No. of Samples in 'simple' Class	145
No. of Samples in 'hard' Class	146

Table 5-9: Best parameters for Random Forest, SVM, and Logistic Regression on the new presentation complexity dataset.

Random Forest	
n_estimators	50
max_depth	5
min_samples_split	6
min_samples_leaf	1
max_features	'sqrt'
max_leaf_nodes	None
SVM	
C	10
kernel	'rbf'
degree	2
gamma	1
Logistic Regression	
penalty	'l2'
solver	'lbfgs'
C	0.1
max_iter	100

Table 5-10: Results of evaluation metrics for Random Forest, SVM, Logistic Regression on the new presentation complexity dataset.

Evaluation Metrics	Random Forest		SVM		Logistic Regression	
	Training Set	Test Set	Training Set	Test Set	Training Set	Test Set
Accuracy	0.985	0.886	0.990	0.886	0.798	0.727
Macro Average Precision	0.986	0.900	0.990	0.894	0.799	0.762
Macro Average Recall	0.985	0.886	0.990	0.886	0.798	0.727
Macro Average F1-Score	0.985	0.885	0.990	0.886	0.798	0.718
ROC AUC	0.999	0.965	1.0	0.956	0.856	0.812

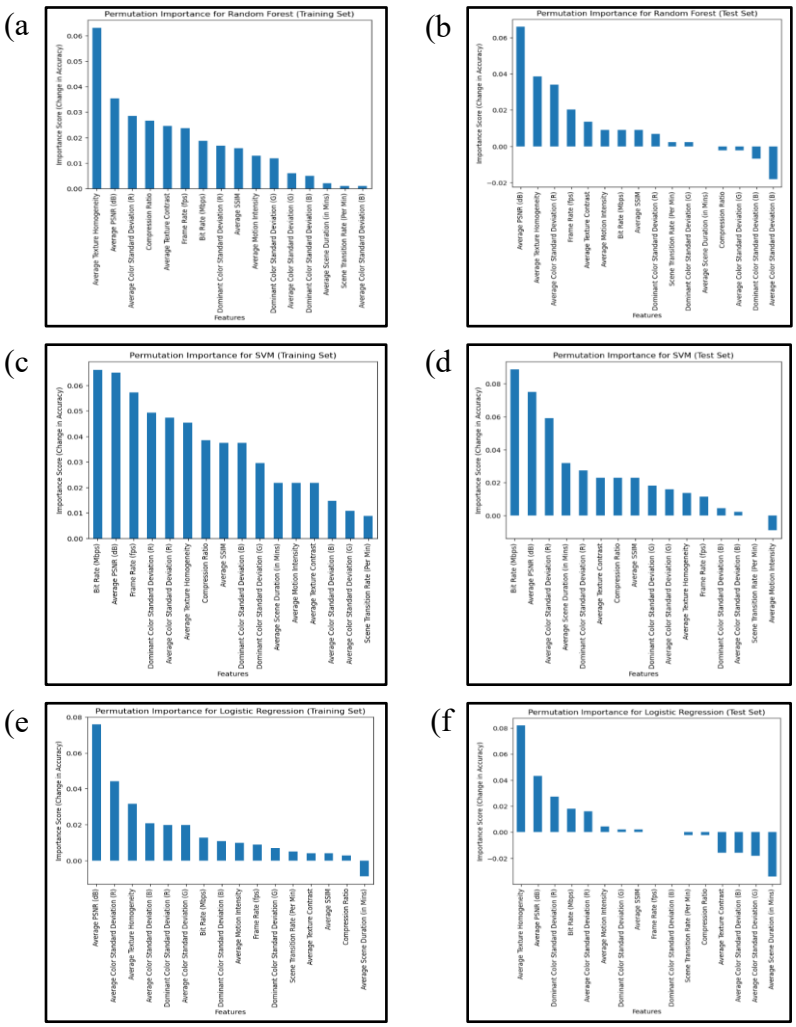

Figure 5-15: Feature importance in Random Forest (a) on the training set, (b) on the test set, SVM (c) on the training set, (d) on the test set, and Logistic Regression (e) on the training set, (f) on the test set of the new presentation complexity dataset.

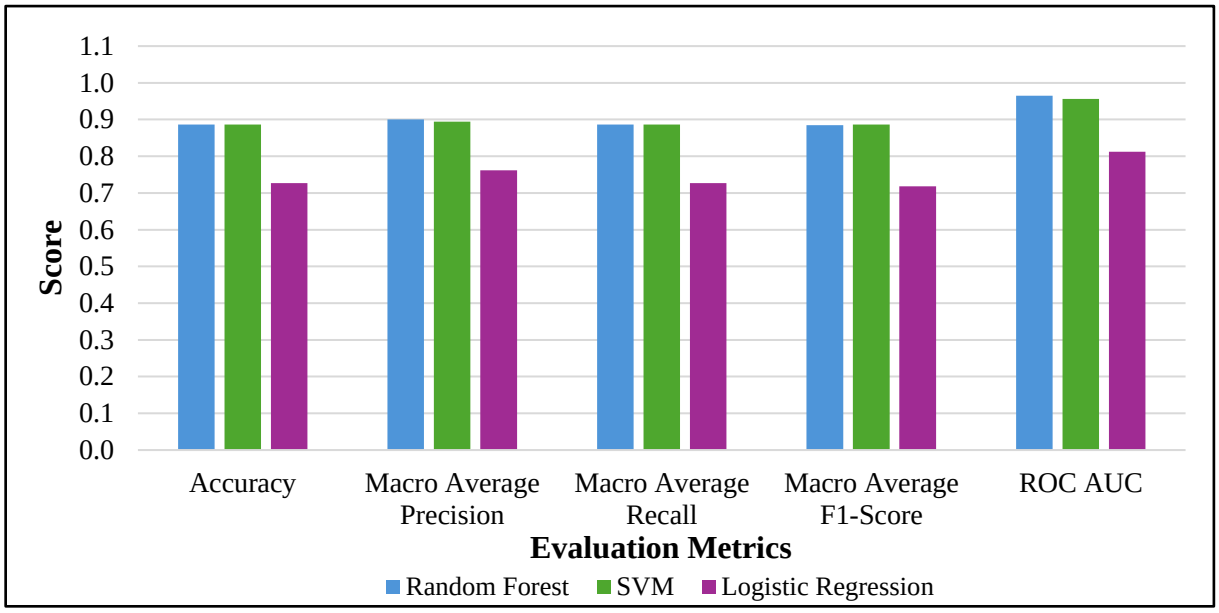

Figure 5-16: Performance comparison of Random Forest, SVM, and Logistic Regression on the test set of the new presentation complexity dataset.

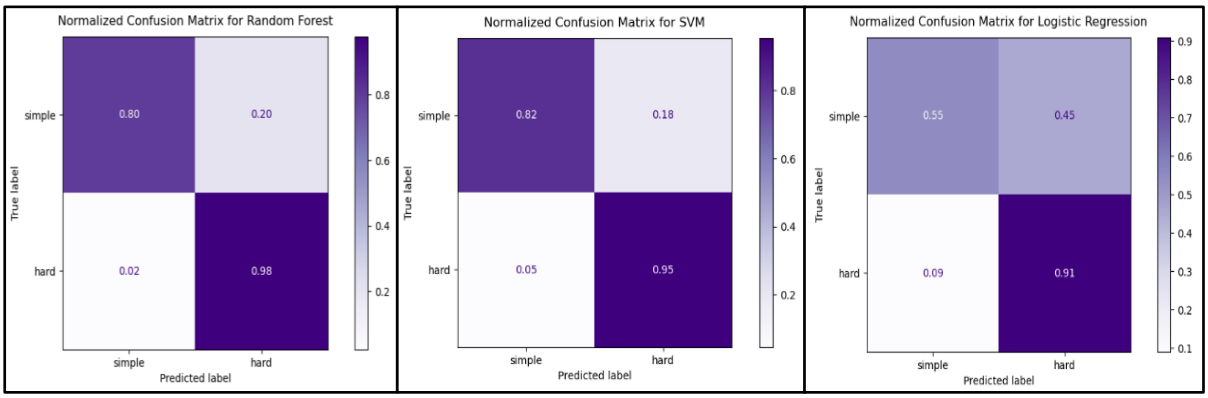

Figure 5-17: Normalized confusion matrix for Random Forest, SVM, and Logistic Regression on the test set of the new presentation complexity dataset.

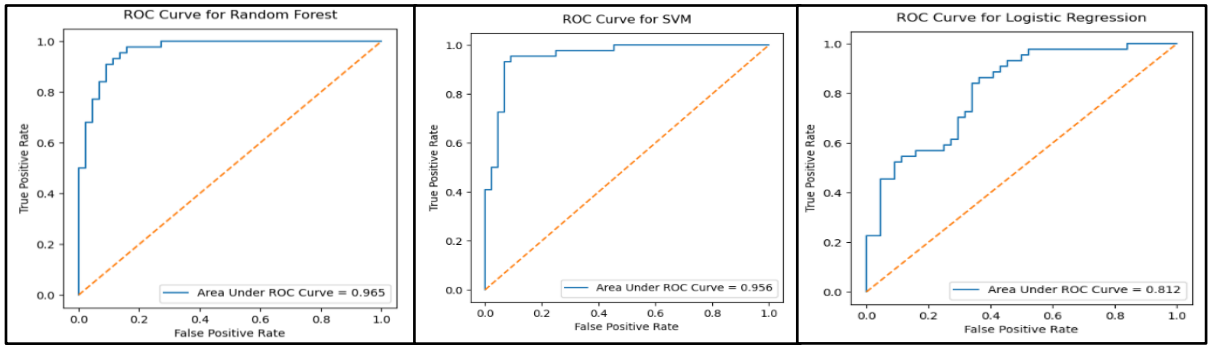

Figure 5-18: ROC curve for Random Forest, SVM, and Logistic Regression on the test set of the new presentation complexity dataset.

5.3.2 Language Complexity

A. Combining Five Classes into Four

This model with five classes demonstrated a significantly poor performance on model testing dataset (refer to section 5.3.2.B), achieving an accuracy of only 35%. To address this issue, we attempted to improve the model's performance by simplifying the classification task by combining the five original grade levels into four broader categories: Kindergarten, Elementary (grades 1-5), Middle Level (grades 6-8), and High Level (grades 9-12). This recategorization aimed to reduce the complexity of the problem and improve the model's ability to distinguish between these broader categories.

After retraining the model with the re-categorized data, we observed a significant improvement in performance. The model achieved an F1-score of 96.3% on the training set and 88.6% on the validation set. This demonstrates the positive impact of recategorizing the grade levels on the model's accuracy and ability to generalize to unseen data. To visualize the training process, the training and validation loss, along with the F1 score curves, can be seen in Figure 5-19.

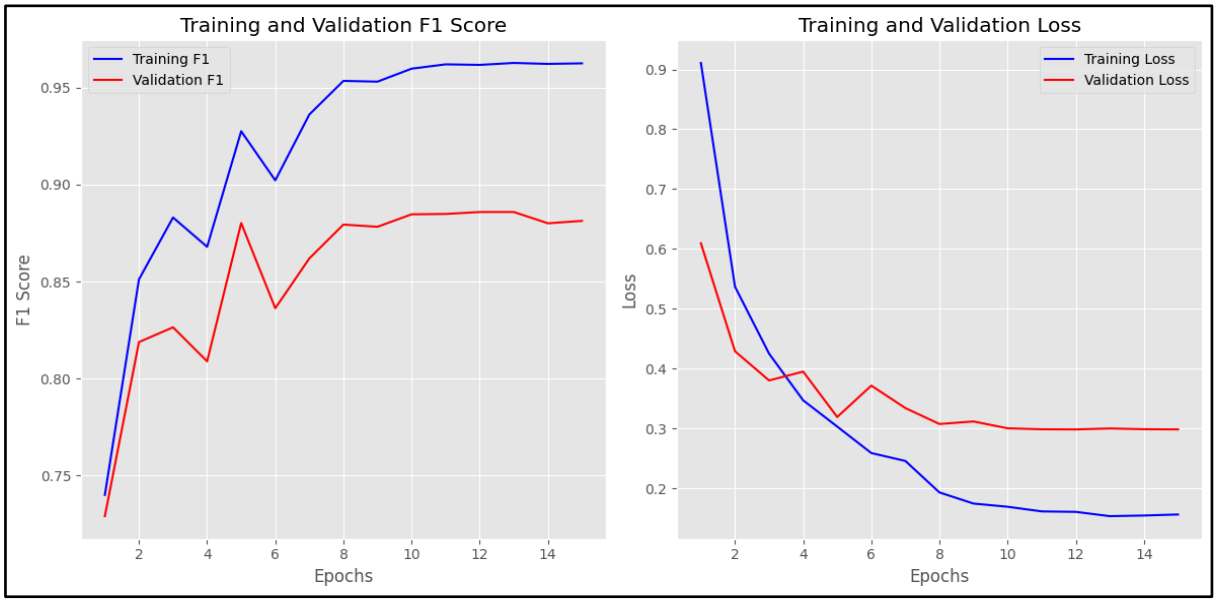

Figure 5-19: Training and validation loss, and F1 score curves after combining into 4-class.

The model was subsequently evaluated on the unseen test set, achieving an F1-score of 87.5% and an accuracy of 88%. The performance of the model on unseen data is visualized through the normalized confusion matrix (Figure 5-20).

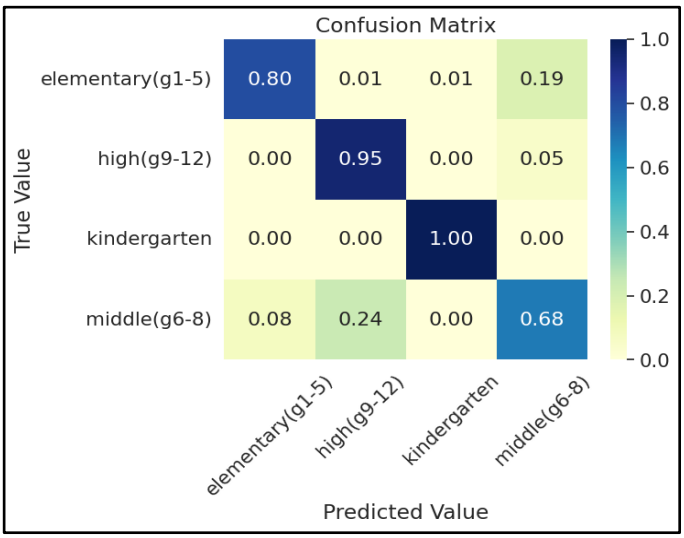

Figure 5-20: Normalized confusion matrix after combining into 4-class.

B. Final Model Testing

To collect data for testing the model, we employed two video annotation methods. Both methods were applied to the 5-class model and to the 4-class model to potentially improve the results.

C. Manual Video Annotation

We first used manual video annotation, where we assigned grade categories to the videos based on our judgment. However, due to unsatisfactory testing results, likely caused by imperfect annotations, we decided to re-annotate the videos for the 4-class model. While this showed some improvement, the results were still not ideal. As a result, we explored ARI-based video annotation as an alternative approach.

D. ARI based Video Annotation

In this approach, we used the Automated Readability Index (ARI) score to determine the grade category for each text extracted from the set of testing videos. After extracting the text, the ARI was calculated for each text to assess its language complexity. Based on the ARI score, each text was assigned to a grade category, except for the kindergarten class, which was annotated manually.

This ARI-based video annotation method was adopted because our model was trained on a dataset sourced online that had been annotated with ARI scores, making the annotations more aligned with how the model would predict the classes. The kindergarten class was excluded from this approach, as the model was trained on separate datasets specifically for kindergarten-level. The testing results for the 5-class model improved with this approach, and for the 4-class model, the results showed even more significant improvement.

E. Text Extraction from Videos

A total of 58 YouTube videos were downloaded using URLs, with a selection from each grade category. The audio from these videos was extracted and converted into text using a multi-step process. First, the audio was extracted using the moviepy library, which saved the audio as .wav files. These audio files were then processed through the OpenAI Whisper large-v3 model. The Whisper model transcribed the audio into text.

F. Testing Results and Discussion

The results with the 5-class system were not satisfactory, so a 4-class system with broader categories was used instead: 'elementary (g1-5)', 'middle (g6-8)', 'high (g9-12)', and 'kindergarten'. This change improved performance, likely because the broader categories gave the model more flexibility, allowing it to better handle variations within each grade group. The results are as stated in (Table 5-11).

The best results were achieved with the 4-class model tested on ARI-based annotation data. However, the high (g9-12) and middle (g6-8) classes still struggled to perform as well as the others. Their performance can be improved in the future by adding more data.

Table 5-11: Model testing results.

Number of Classes	Video Annotation Approach	Normalized Confusion Matrix	Accuracy	Macro Average Precision	Macro Average Recall	Macro Average F1-score
5	Manual	(Figure 5-21)	35%	42%	35%	29%
5	ARI based	(Figure 5-22)	57%	65%	58%	59%
4	Manual	(Figure 5-23)	64%	69%	60%	60%
4	ARI based	(Figure 5-24)	71%	76%	66%	68%

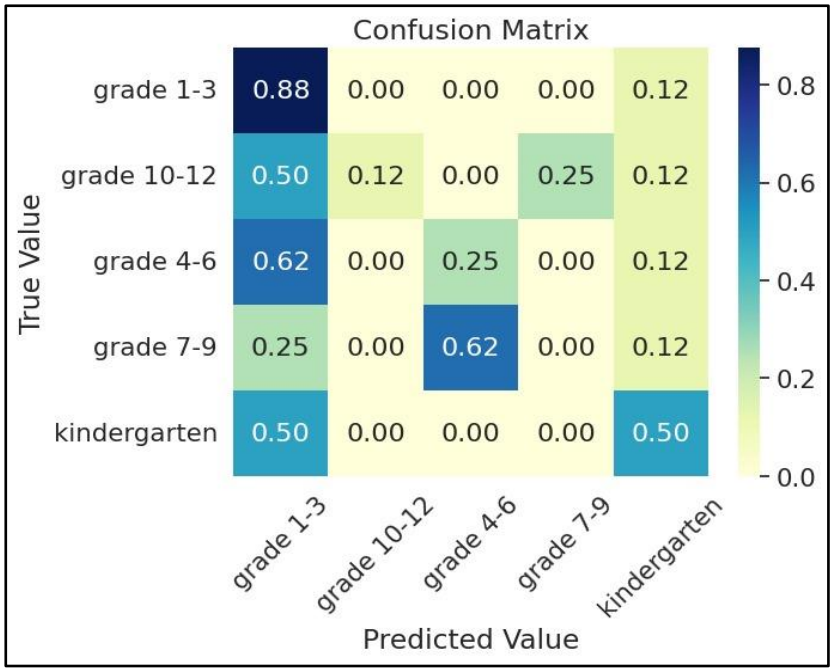

Figure 5-21: Normalized confusion matrix for 5 class using manual annotation.

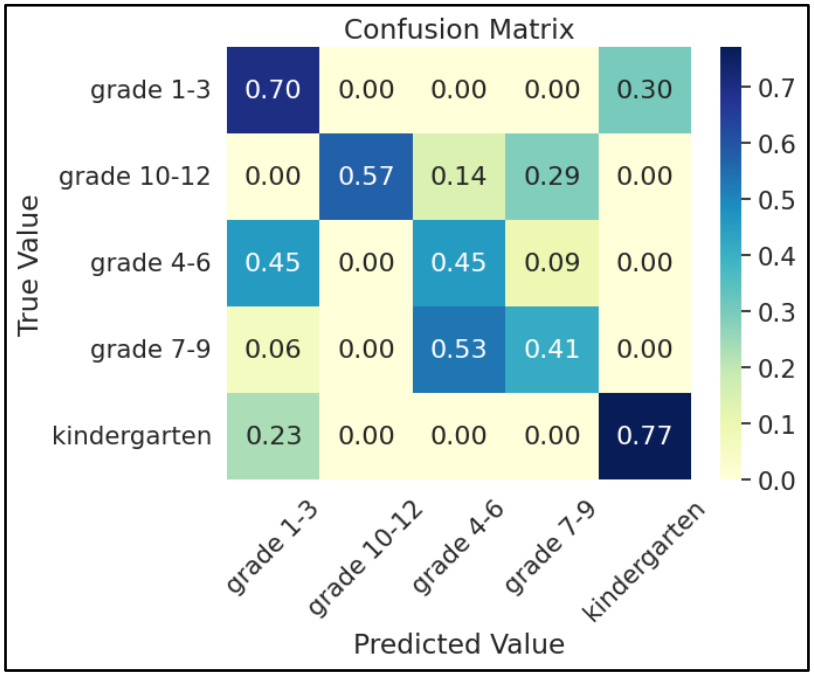

Figure 5-22: Normalized confusion matrix for 5 class using ARI based annotation.

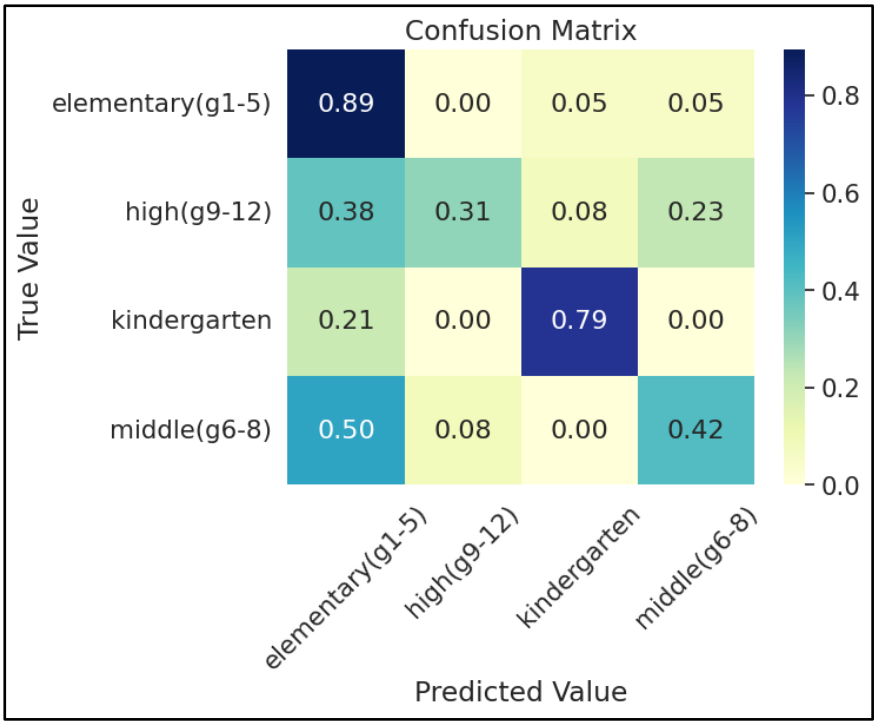

Figure 5-23: Normalized confusion matrix for 4 class using manual annotation.

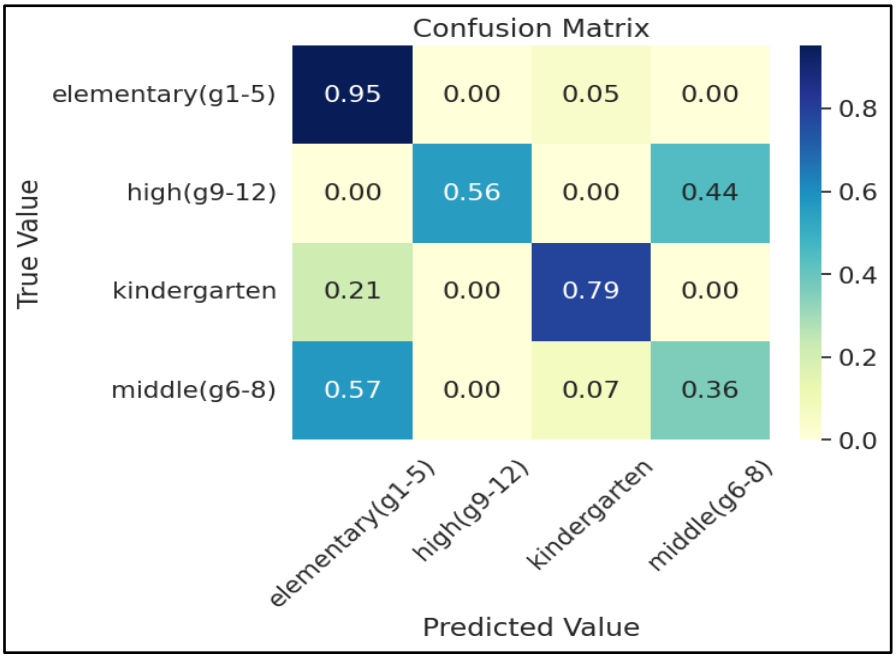

Figure 5-24: Normalized confusion matrix for 4 class using ARI based annotation.

In conclusion, by simplifying grade-level classification into broader categories (Kindergarten, Elementary, Middle, and High School), we improved the model's performance. Achieved strong results, with an F1-score of 96.3% on the training set and 87.5% on the test set. Additionally, we performed model testing using text extracted from YouTube videos annotated by the ARI score, achieving an F1-score of 69%. Future improvements could enhance the model testing accuracy, particularly for the middle and high school categories.

CHAPTER 6: Web Development

6.1 Front-End

By referring to our User Interface (UI) design (Figure 4-4) on Figma, which was completed during the Junior Project, we developed the front-end of the ACES web application using HTML, CSS, and JavaScript. With HTML and CSS, the website's structural and visual appearance were defined. JavaScript was utilized to validate the checker and feedback forms' input fields in the Video Appropriateness Checker and Footer sections of the website, respectively. Below describes the included validations:

Checker form: Do not allow the user to submit the form if:

- No video file is provided
- Given video file format is invalid.
- No appropriateness evaluation criteria is selected.

Feedback form: Do not allow the user to submit the form if:

- No feedback is provided, including whitespaces.

Figure 6-1 displays some of the instances of validation on the website.

Initially, the website was constructed to suit the screen size of standard laptops. Then, using the media query method in CSS, the website's content appearance was modified to adapt to browser widths of at most 938px and 630px, thereby enhancing the responsiveness of the website. These browser widths were chosen because that is when the content layout changed to display an unorganized design. The designs for both browser widths represent the same layout, with the only difference being the size of the elements. Figure 6-2 illustrates the front-end implementation of ACES, while Figure 6-3 shows its responsive design.

In addition, the pytubefix video downloading library, which was used to download YouTube videos, is not working anymore. All the other available libraries are also generating errors. Hence, we were unable to implement a function on the ACES back-end to automatically download videos using the URL inputs from users. Therefore, we included only the video file input feature on the ACES front-end and removed the video URL input feature.

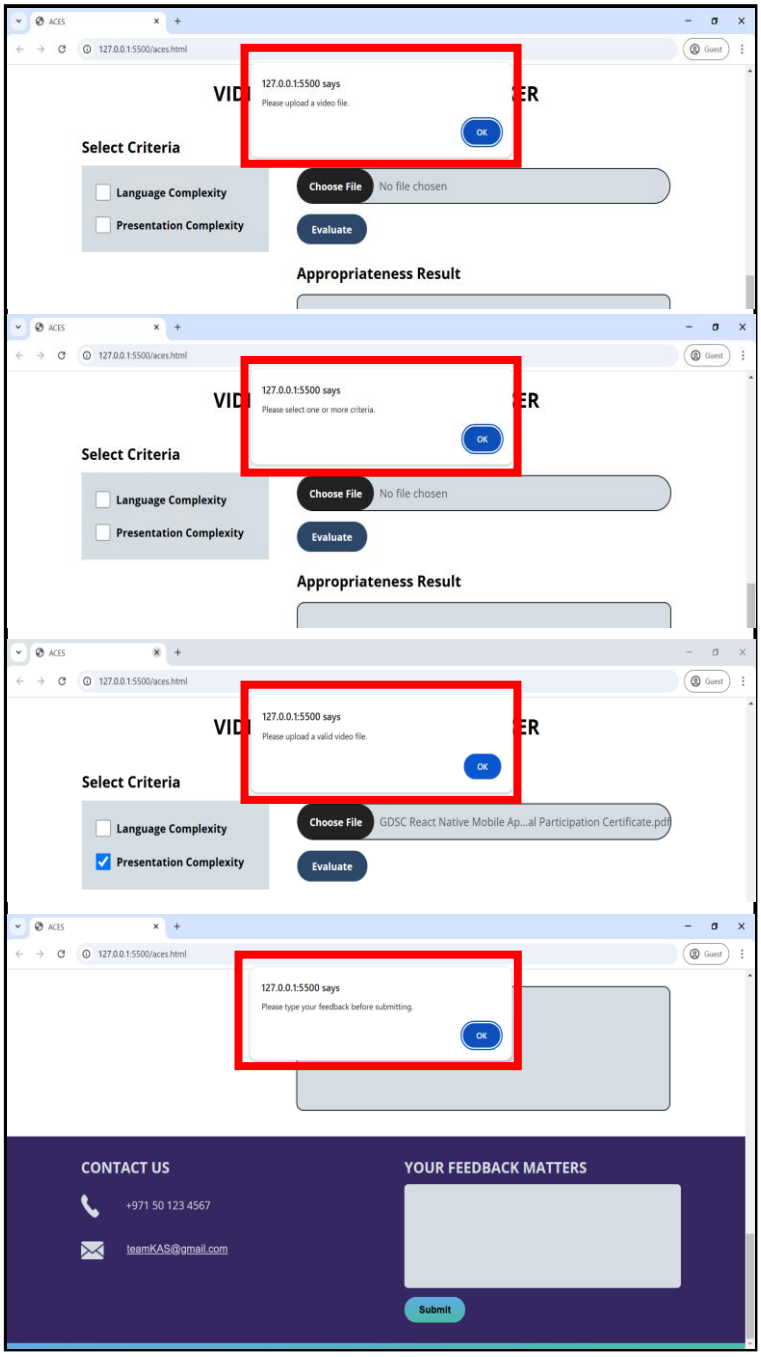

Figure 6-1: Form validation (red squares) implemented using JavaScript.

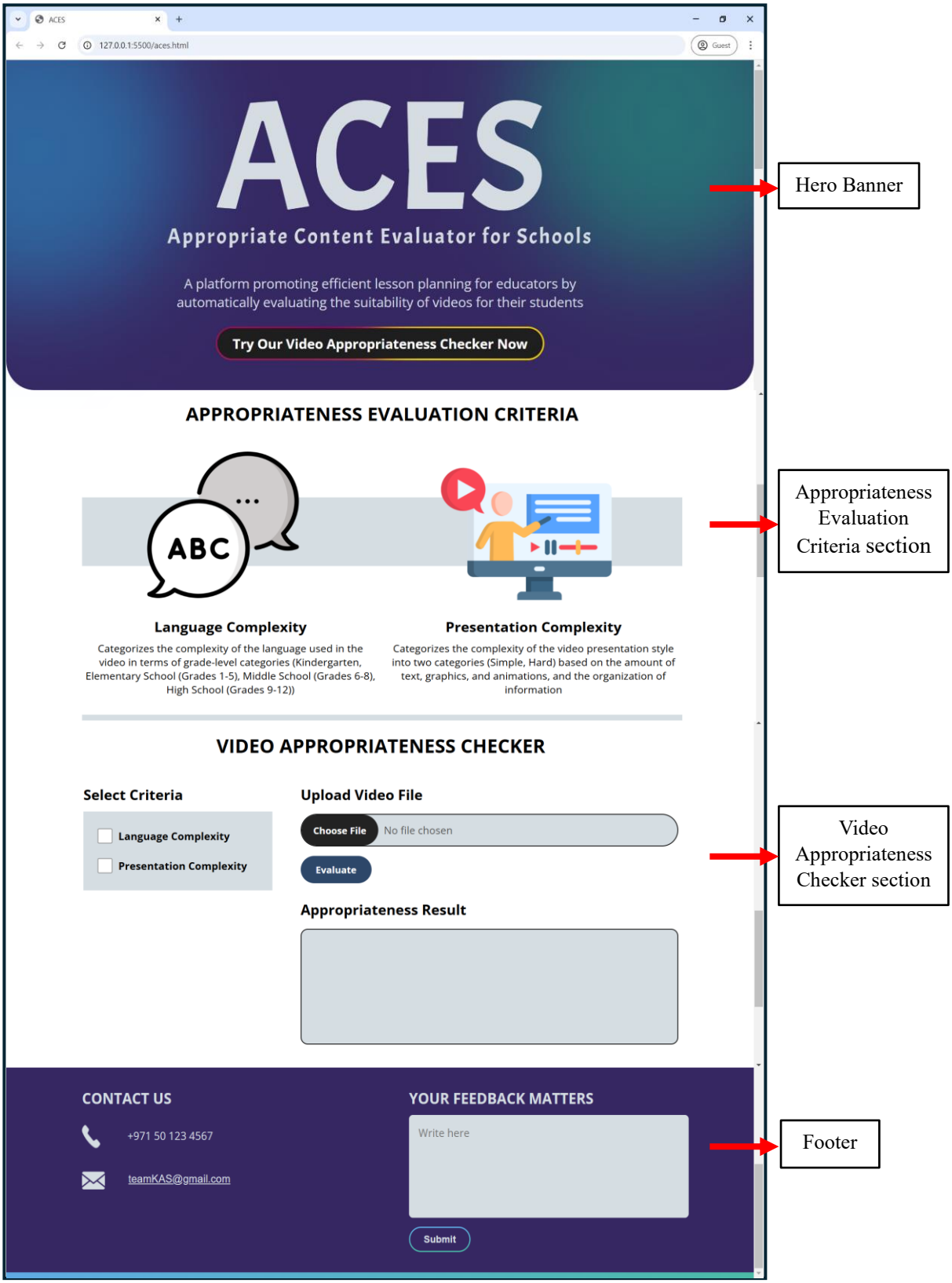

Figure 6-2: Front-end implementation of the ACES web application.

(a) (b)

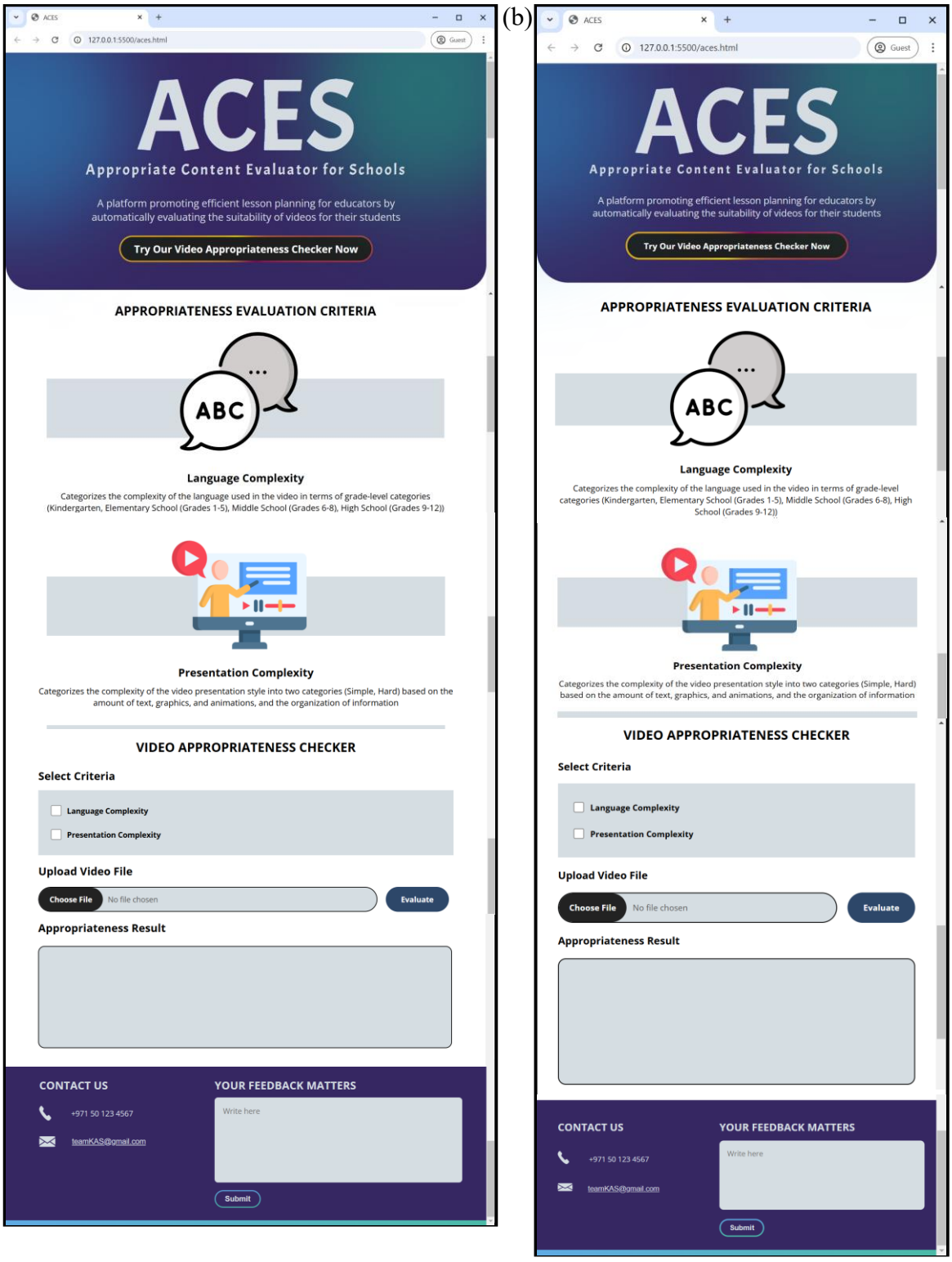

Figure 6-3: Responsive design of ACES website for browser widths (a) <= 938px (b) <= 630px.

6.2 Back-End

Flask is a Python-based web application framework, allowing developers to build the back-end of their web applications. One of the main features of Flask is its micro web framework, which provides simplicity, scalability, and extensibility in application development [41]. We utilized Flask to develop the back-end of the ACES website, where both the machine learning and deep learning models of Presentation Complexity and Language Complexity have been integrated to perform video appropriateness evaluation. However, due to the time constraint, we were unable to address the following issues that affect users' experience:

1. Flask reloads the web page before showing the response. This resets the form input fields to their default values, making the user inputs disappear and displaying only the output. In other words, the output is not added to the website dynamically.
2. The user cannot reload the web page after the video appropriateness checker or feedback form submission. If the user attempts to reload, the browser will display a 'Confirm Form Resubmission' message and approving this will repeat the video processing or feedback submission.
3. There is no database implementation to store the users' feedback.

CHAPTER 7: Challenges Faced

The following are the challenges we faced during this project:

- Creating the presentation complexity dataset was time-consuming as it required manual data collection including searching for suitable videos as well as downloading the collected videos.
- Selecting an appropriate language complexity dataset with existing annotations was a complex task.
- Annotating manually collected videos was challenging as we were unable to receive experts' help.
- Determining the optimal values for the hyperparameter tuning of language complexity model to achieve superior results posed a significant difficulty as it was done manually.
- Developing the back-end of the web application was difficult as we faced version compatibility problems between Flask and the libraries used for model implementation.

CHAPTER 8: Conclusion and Future Work

Our work addresses the increasing reliance of teachers on videos as educational resources. By analysing both presentation complexity and language complexity, our work aims to simplify the process of selecting suitable videos for students.

To assess presentation complexity, we employed machine learning techniques on a dataset of annotated educational videos mainly sourced from YouTube. We analysed the presentation complexity using different machine learning algorithms, among which, Random Forest showed better performance with an accuracy of 88.6%, precision of 90.6%, recall of 88.6%, F1-score of 88.5%, and ROC AUC of 0.965.

To assess language complexity based on grade level, we settled with a four-class model system instead of five. We successfully implemented a four-class classification model using the Distil BERT framework. The model achieved an F1-score of 96.3% on the training set and 88.6% on the validation set and 87.5% on testing set. We carried out separate model testing using text from videos annotated by ARI score which achieved F1-score of 69%.

Additionally, we designed and implemented a responsive front-end for the ACES web application, while on the back end, we successfully integrated the two AI models and created a system to process user inputs and evaluate video appropriateness using the trained models.

Our findings demonstrate the effectiveness of our approach in evaluating both presentation and language complexity within educational videos. By providing teachers with insights into these factors, our work can assist in selecting videos that align with student's comprehension levels and learning styles

Due to time constraints, we were unable to implement a database to store video metadata, appropriateness results, and user feedback, which would have enhanced model learning and back-end functionality. The lack of annotated grade-level video datasets online limited our ability to gather enough videos for model testing, affecting the reliability of the results. Additionally, we were unable to conduct comprehensive software verification and usability testing.

Looking ahead, we plan to address these challenges by collecting and annotating more videos, implementing the database, and performing thorough testing to improve both model

performance and user experience. Furthermore, we aim to expand the web application by integrating additional evaluation criteria such as Mature Content, Clarity of Image and Sound, and Inclusion of Real-Life Examples, which will enhance its overall functionality and effectiveness.

CHAPTER 9: References

[1] Perienen, A., Meera, G., Teenah, J., & Preetamsingh Dookhun. *Confused which educational video to choose? appropriateness of YouTube videos for instructional purposes-making the right choice*

[2] *How to use video content safely in the classroom.* (2023). https://edexec.co.uk/how-to-use-video-content-safely-in-the-classroom/

[3] Alam, I., Uddin, A., & Khusro, S. (2022). *Age-based ranking of YouTube videos for improved parental controls in smart TV environment* IEEE. doi:10.1109/icetecc56662.2022.10069109

[4] *What is computer vision: Applications, benefits and how to learn it.* (2023). https://www.simplilearn.com/computer-vision-article

[5] Alexander S. Gillis. (2024). *Natural language processing (NLP)* .https://www.techtarget.com/searchenterpriseai/definition/natural-language-processing-NLP

[6] *How clarifai content moderation works.*https://www.clarifai.com/solutions/content-moderation

[7] *Azure AI content safety documentation.*https://learn.microsoft.com/en-us/azure/ai-services/content-safety/

[8] *Safe video search.*https://www.safesearchkids.com/safe-video-search/

[9] *Common sense.*https://www.commonsense.org/

[10] Arshad, M., Yousaf, M. M., & Sarwar, S. M. (2023). *Comprehensive readability assessment of scientific learning resources* Institute of Electrical and Electronics Engineers (IEEE). doi:10.1109/access.2023.3279360

[11] Islam, M. S., Rony, M. A. T., Saha, P., Ahammad, M., Nazmul Alam, S. M., & Saifur Rahman, M. (2023). *Beyond words: Unraveling text complexity with novel dataset and A classifier application* IEEE. doi:10.1109/iccit60459.2023.10441159

[12] Sha, H., & Yep, T. (2018). *CS 229 project report: Text complexity (natural language)*

[13] *BBFC classification guidelines.* https://www.bbfc.co.uk/about-classification/classification-guidelines

[14] Juxtopposed. (2023). *World's shortest UI/UX design course.* https://youtu.be/wIuVvCuiJhU?si=VvvxStz9G2NLgZYL

[15] *The ultimate guide to user flow diagram.* (2022). https://bootcamp.uxdesign.cc/the-ultimate-guide-to-user-flow-diagram-b108d7de10d

[16] Adrian Twarog. (2021). *UI / UX design tutorial – wireframe, mockup & design in figma .* https://youtu.be/c9Wg6Cb_YlU?si=mkONBrzpjWwIYWFE

[17] Aniruddha Bhandari. (2024). *Feature scaling: Engineering, normalization, and standardization (updated 2025).* https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/

[18] Max Tingle. (2019). *Preventing data leakage in your machine learning model.* https://towardsdatascience.com/preventing-data-leakage-in-your-machine-learning-model-9ae54b3cd1fb

[19] Anchal Agarwal. (2021). *OpenCV – the gunnar-farneback optical flow.* https://www.geeksforgeeks.org/opencv-the-gunnar-farneback-optical-flow/

[20] Girish Ajmera. (2024). *Feature extraction of images using GLCM (gray level cooccurrence matrix).* https://medium.com/@girishajmera/feature-extraction-of-images-using-glcm-gray-level-cooccurrence-matrix-e4bda8729498

[21] NeuralNine. (2022). *Extracting dominant colors from images in python.* https://www.youtube.com/watch?v=nEYap1mupUQ

[22] *Capture bit rate.* https://workflow.frame.io/guide/bit-rate

[23] Peter Forret. *Video filesize calculator.* https://toolstud.io/video/filesize.php

[24] nimpy. (2024). *Calculating peak signal-to-noise ratio (PSNR) between two images.* https://gist.github.com/nimpy/5b0085075a54ba2e94f2cfabf5a98a57

[25] *Peak signal-to-noise ratio as an image quality metric.* (2024). https://www.ni.com/en/shop/data-acquisition-and-control/add-ons-for-data-acquisition-and-control/what-is-vision-development-module/peak-signal-to-noise-ratio-as-an-image-quality-metric.html?srsltid=AfmBOop3aSzAi592FzTILV01n0SK792ANrRbMi6fVrMMKeUH57BbKpUX

[26] Adrian Rosebrock. (2017). *Image difference with OpenCV and python.*https://pyimagesearch.com/2017/06/19/image-difference-with-opencv-and-python/

[27] Ammar Ali. (2024). *How to calculate average color of image in OpenCV.*https://www.delftstack.com/howto/python/opencv-average-color-of-image/

[28] Castellano, B. PySceneDetect [Computer software]. https://github.com/Breakthrough/PySceneDetect

[29] van der Walt, S. J., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., Yu, T., & the scikit-image contributors. (2014). scikit-image: image processing in Python. PeerJ, 2, e453. https://doi.org/10.7717/peerj.453

[30] Taylor Jensen. (2022). *Feature importance for any model using permutation.*https://medium.com/@T_Jen/feature-importance-for-any-model-using-permutation-7997b7287aa

[31] Arpita. (2024). *Machine learning explainability using permutation importance.*https://www.geeksforgeeks.org/machine-learning-explainability-using-permutation-importance/

[32] *common lit external dataset 2021*. (2021, August 1). Kaggle. https://www.kaggle.com/datasets/sayantankirtaniya/newone?select=all_data.csv

[33] Scrosseye. (n.d.). *GitHub - scrosseye/CLEAR-Corpus: Repository for the CommonLit Ease of Readability Corpus*. GitHub. https://github.com/scrosseye/CLEAR-Corpus/tree/main

[34] *English Nursery Rhymes*. (2023, December 15). https://www.kaggle.com/datasets/terencebroad/english-nursery-rhymes

[35] Aydin, A. (2023, October 5). 1 — Text Preprocessing Techniques for NLP - Aysel Aydin - Medium. *Medium*. https://ayselaydin.medium.com/1-text-preprocessing-techniques-for-nlp-37544483c007

[36] Li, Z., Ding, H., & Zhang, S. (2023). Cross-corpus readability compatibility assessment for english texts Institute of Electrical and Electronics Engineers (IEEE). doi:10.1109/access.2023.3315834

[37] Benjamin Consolvo. (2024). Automatic Speech Recognition Using OpenAI Whisper without a GPU. *Medium*. https://medium.com/intel-analytics-software/automatic-speech-recognition-using-openai-whisper-without-a-gpu-9d316a93860a

[38] Sanh, V. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

[39] *DistilBERT.*https://huggingface.co/transformers/v3.0.2/model_doc/distilbert.html

[40] *Youtube to MP4 converter.*https://en.y2mate.is/u2ZQt/youtube-to-mp4.html

[41] *What is flask python.*https://pythonbasics.org/what-is-flask-python/