



**RAJALAKSHMI
ENGINEERING COLLEGE**

An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

**ELOQUORA
IMPROVING DEBATE & DISCUSSION SKILLS:
AI - BASED ARGUMENT MINING & SPEECH
EVALUATION**

Submitted by
SRIHARINI S (221501141)
SWADHI R (221501155)

AI19643 FOUNDATIONS OF NATURAL LANGUAGE PROCESSING

Department of Artificial Intelligence and Machine Learning

Rajalakshmi Engineering College, Thandalam



BONAFIDE CERTIFICATE

NAME

ACADEMIC YEAR.....SEMESTER.....BRANCH.....

UNIVERSITY REGISTER No.

Certified that this is the bonafide record of work done by the above students in the Mini Project titled **"ELOQUORA - IMPROVING DEBATE & DISCUSSION SKILLS: AI - BASED ARGUMENT MINING & SPEECH EVALUATION"** in the subject **AI19643 FOUNDATIONS OF NATURAL LANGUAGE PROCESSING** during the year **2024 - 2025**.

Signature of Faculty – in – Charge

Submitted for the Practical Examination held on _____

INTERNAL EXAMINER

EXTERNAL EXAMINER

ABSTRACT

In an era where communication and critical thinking are indispensable, the development of persuasive speaking and debate skills has become a vital component of personal and professional growth. This project presents Eloquora, an AI-powered debate and discussion partner designed to assist users in enhancing their speaking abilities through interactive, real-time feedback. Leveraging the capabilities of Natural Language Processing (NLP), speech recognition, and large language models (LLMs), the system offers an immersive experience where users can engage in discussions via text or speech. Key technologies integrated into the platform include OpenAI's Whisper for accurate speech-to-text transcription (achieving a Word Error Rate of 0.12 and Character Error Rate of 0.05) and Gemma (via Ollama) for generating contextually relevant and logically sound responses. The argument mining module, based on an LSTM architecture, demonstrates a classification accuracy ranging from 89% to 92%. The application supports a chatbot-like interaction mode, allowing users to converse continuously until they choose to exit. Additionally, the system evaluates user input for tone, emotional expression (using models like BERT-Go-Emotion), clarity, and argumentative structure, offering constructive suggestions for improvement. This encourages not only effective articulation but also mindful listening and reasoning. Eloquora is particularly beneficial for students, professionals, and individuals preparing for interviews, debates, or public speaking engagements. By delivering personalized, non-judgmental feedback in a conversational format, it demonstrates the potential of artificial intelligence as a practical tool for soft-skill development and lifelong learning. This initiative highlights how emerging AI technologies can be meaningfully applied in education and communication training, offering scalable, intelligent, and performance-validated solutions that foster confident and articulate expression.

Keywords:

AI Debate Coach, Natural Language Processing, Speech Recognition, Communication Skills, Argument Evaluation, Language Model

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT	
1.	INTRODUCTION	1
2.	LITERATURE REVIEW	3
3.	SYSTEM REQUIREMENTS	8
	3.1 HARDWARE REQUIREMENTS	8
	3.2 SOFTWARE REQUIREMENTS	8
4.	SYSTEM OVERVIEW	9
	4.1 EXISTING SYSTEM	9
	4.1.1 DRAWBACKS OF EXISTING SYSTEM	9
	4.2 PROPOSED SYSTEM	10
	4.2.1 ADVANTAGES OF PROPOSED SYSTEM	10
5.	SYSTEM IMPLEMENTATION	12
	5.1 SYSTEM ARCHITECTURE	12
	5.2 SYSTEM FLOW	13
	5.3 LIST OF MODULES	14
	5.4 MODULE DESCRIPTION	14
6.	RESULT AND DISCUSSION	18
7.	APPENDIX	22
	SAMPLE CODE	22
	OUTPUT SCREENSHOTS	40
	REFERENCES	42

CHAPTER 1

INTRODUCTION

Effective communication has become one of the most essential skills in the modern world—whether in academic institutions, professional workplaces, or even informal social settings. The ability to articulate ideas clearly, respond thoughtfully, and engage in meaningful conversations or structured debates is central to both personal growth and professional success. As the demands of the 21st century evolve, so does the importance of developing strong soft skills such as public speaking, active listening, persuasive reasoning, and critical thinking. However, despite this growing need, many individuals still find it difficult to cultivate these abilities due to a lack of structured training environments, timely feedback, or regular opportunities for practice.

For most people, traditional debate coaching or communication workshops can be expensive, time-consuming, or simply inaccessible. This lack of practical support creates a gap between the skills people want to improve and the resources available to them. As someone preparing for a job interview, delivering a classroom presentation, or simply trying to participate more confidently in discussions, it can be frustrating not having a consistent way to practice or receive helpful feedback. In response to this very real challenge, the project introduces Eloquora, an AI-powered discussion and debate coach designed to support users in improving their speaking and reasoning skills through interactive, conversational learning.

Eloquora harnesses the power of modern Artificial Intelligence technologies, particularly advancements in Natural Language Processing (NLP), speech recognition, and large language models. At its core, the system creates an environment where users can practice formulating arguments, expressing their thoughts, and engaging in dialogue with an AI that provides relevant, logical, and supportive responses. The interaction is designed to be natural, continuous, and user-friendly—allowing individuals to focus on their communication without being overwhelmed by technical complexity.

Technically, the platform integrates multiple AI components. The Whisper model is used for accurate speech-to-text transcription, enabling users to speak naturally using a

microphone. The Gemma language model, powered via the Ollama framework, generates thoughtful and context-aware responses based on user input. Whether the user chooses to type or speak, the system engages in an ongoing conversation that only ends when the user says commands like “bye,” “done,” “quit,” or “exit.” This continuous loop mirrors the experience of chatting with a human coach or mentor, making practice sessions feel more intuitive and engaging.

Ultimately, the objective of this project is to develop an intelligent, real-time conversational system that helps users speak with clarity, confidence, and logical consistency. By combining the strengths of speech recognition and generative AI, Eloquora demonstrates how technology can be harnessed not just to automate tasks, but to actively support human learning and communication. This report presents a detailed look into the system’s design, implementation, and real-world significance in shaping the future of communication training.

CHAPTER 2

LITERATURE REVIEW

[1] **Title:** Logical Fallacy Annotation in Natural Language Arguments

Author: Joel J. Wallenberg, Robert West, et al. (2022)

This paper introduces a dataset and methodology for detecting logical fallacies in natural language arguments, a crucial component of discourse understanding and argument analysis. The authors present a corpus annotated with 13 types of logical fallacies, curated from Reddit’s “Change My View” discussions. The dataset comprises over 30,000 annotated samples, manually labeled by trained linguists. They experimented with both classical machine learning models and modern transformers such as RoBERTa and BERT for multi-label classification. Their best model achieved a macro F1 score of 0.47. Although this is promising, limitations include difficulty in identifying subtle fallacies and a dataset skew toward certain fallacy types. The authors also released code and data publicly, encouraging further research in fallacy-aware dialogue systems.

[2] **Title:** LLM-based Rewriting of Inappropriate Argumentation using Reinforcement Learning from Machine Feedback

Author: He Zhao et al. (2024)

This recent study explores how large language models (LLMs) can improve argumentation quality by rewriting inappropriate arguments using reinforcement learning (RL) guided by machine feedback. The system is trained on a corpus of 200,000 political and controversial debates, and leverages LLMs like GPT-3.5 to generate revised arguments. The machine feedback component evaluates toxicity, logical coherence, and sentiment to generate scalar rewards for RL optimization. The model significantly outperformed baselines in terms of coherence and non-toxicity, achieving a 13% improvement in human evaluations. However, challenges remain in aligning LLM outputs with subtle cultural or emotional nuances not captured by automated feedback.

[3] **Title:** SOUL: Towards Sentiment and Opinion Understanding of Language

Author: Yihao Feng et al. (2023)

The SOUL framework presents a comprehensive benchmark for sentiment and opinion understanding using a multi-dimensional annotation schema. The dataset includes over 40,000 English sentences covering sentiment polarity, opinion target, and speaker intent. The authors employed pre-trained transformers like DeBERTa and T5, fine-tuned on the benchmark tasks. Results showed an improvement of 9–12% in accuracy over traditional sentiment models. Notably, SOUL emphasizes disentangling nuanced emotions and speaker attitudes. A key limitation is the lack of real-time adaptability and the model's performance drop on domain-specific or sarcastic text inputs. Nonetheless, the paper contributes significantly to sentiment-aware NLP systems.

[4] **Title:** Introducing Syntactic Structures into Target Opinion Word Extraction with Deep Learning

Author: Liheng Xu et al. (2022)

This paper addresses the challenge of extracting target-opinion pairs in sentiment analysis using syntactic dependency structures. The proposed model, SST4, integrates BiLSTM with syntactic cues derived from dependency parsers. Trained on SemEval and Amazon Review datasets, the approach significantly improved precision and recall by over 10% compared to baseline LSTM and CRF models. The method effectively identifies nuanced relationships between opinionated expressions and their targets. However, its reliance on high-quality syntactic parsers may limit performance on informal or noisy text. The study bridges syntax and deep learning for fine-grained opinion mining.

[5] **Title:** An End-to-End Neural Argument Mining System for Essay Scoring

Author: Manfred Stede et al. (2022)

This paper proposes an end-to-end neural pipeline for mining argumentative structures from student essays, aiming to support automated essay scoring. The system uses a joint BiLSTM-CRF architecture for argument component identification (claim, premise,

rebuttal) and relation detection. Trained on the UKP Essays corpus (25,000 essays), the model achieved an F1 score of 0.71 for argument classification and 0.64 for link prediction. A notable feature is its robustness in handling free-form, unstructured student writing. While promising, the model sometimes misclassifies rhetorical devices and struggles with implicit premises. Future work could integrate discourse-level cues and human-in-the-loop feedback.

[6] **Title:** Detecting Argumentative Discourse Structures in Social Media

Author: Xiaoming Yu et al. (2023)

This study explores argument mining in social media, particularly Twitter debates, using a graph-based attention model. The dataset comprises 100K annotated tweets from online debates on climate change, gun laws, and vaccination. The model employs GAT (Graph Attention Networks) to identify argument components and their relationships. It achieved a macro-F1 score of 0.69, outperforming transformer baselines on this noisy, short-text dataset. However, limitations include inconsistent labeling due to tweet length constraints and sarcastic tones. The model's use of conversation thread structure as contextual input marks a novel approach to argument analysis in real-world dialogue.

[7] **Title:** Real-time Emotion-Aware Text-to-Speech System using DeepSpeech and BERT

Author: A. Patil et al. (2023)

This paper presents a real-time emotion-aware TTS system that uses BERT to detect sentence-level emotions and modulate voice synthesis accordingly via DeepSpeech and Tacotron2. The model is trained on the IEMOCAP dataset with over 12,000 emotion-labeled utterances. It achieves 83% emotion classification accuracy and naturalness scores of 4.1/5 in user evaluations. The system can embed prosody based on emotion labels like anger, joy, or sadness. However, real-time deployment remains constrained by inference latency and TTS model complexity. It contributes significantly to human-like interaction in voice AI systems.

[8] **Title:** Sentiment and Sarcasm Detection in Multilingual Text using Dual-LSTM Architecture

Author: I. Ghosh et al. (2022)

This paper proposes a multilingual sentiment and sarcasm detection model using a dual BiLSTM with shared attention layers. Tested on Hindi-English and Tamil-English code-mixed datasets from social media, the model achieved 82% accuracy in sentiment classification and 74% in sarcasm detection. The architecture leverages subword embeddings and transfer learning. A unique contribution is its handling of implicit sentiment shifts caused by sarcasm. Limitations include reduced accuracy on idiomatic expressions and limited availability of high-quality multilingual sarcasm datasets. It provides strong insights for multilingual and nuanced sentiment analysis.

[9] **Title:** A Reinforcement Learning-based Framework for Text Rewriting to Improve Argument Clarity

Author: M. Lee et al. (2023)

This work uses reinforcement learning to improve argument clarity by rewriting ambiguous or convoluted sentences. The system is trained on annotated political discourse from Reddit and Debatepedia, using reward signals based on argument quality metrics such as specificity and coherence. Results show a 16% improvement in readability and coherence compared to supervised baselines. The model leverages GPT-2 as the generator and uses feedback from a separate critique model. A limitation is the potential over-simplification of complex arguments. Nonetheless, this framework aligns well with feedback-driven AI communication coaching tools.

[10] **Title:** Multimodal Argument Quality Estimation in Debate Videos

Author: T. Nakamura et al. (2022)

This study investigates the assessment of argument quality using multimodal inputs (text, audio, visual cues) in debate videos. The authors created a dataset of 1,200 debate clips annotated for persuasiveness, logical consistency, and speaker confidence. Using a fusion of BERT, ResNet (visual), and MFCC (audio) features, the model achieved a correlation

of 0.78 with human-rated argument quality scores. A limitation is the requirement for high-quality video and audio data, which restricts scalability. Nevertheless, this work offers a foundation for holistic AI evaluation in debate coaching platforms like Eloquora.

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 HARDWARE REQUIREMENTS

- CPU: Intel Core i5 or better
- GPU: AMD Ryzen 5 or higher
- Hard Disk: 256GB SSD
- RAM: 8GB or more
- Audio Input: Built-in or external microphone
- Audio Output: Built-in or external speakers/headphones
- Power Supply: Uninterruptible Power Supply (UPS) for continuous operation

3.2 SOFTWARE REQUIREMENTS

- Programming Environment: Python 3.9 or above
- IDE: Visual Studio Cod (v1.60+) or Jupyter Notebook (v6.0+)
- Operating System: Windows 10 or higher
- Virtual Environment :venv310
- Whisper (by OpenAI) : base or higher
- Ollama: gemma:2b
- gTTS : version 2.3.2
- NLTK : version 3.8.1
- Data Analysis Tools: Pandas (v1.1+) and Matplotlib (v3.3+) for data visualization

CHAPTER 4

SYSTEM OVERVIEW

4.1 EXISTING SYSTEM

The current landscape of communication training and debate assistance systems includes a range of tools such as e-learning platforms, rule-based chatbots, public speaking clubs (e.g., Toastmasters), and mobile coaching apps. These systems utilize technologies like basic Natural Language Processing (NLP) for grammar correction, speech-to-text conversion for transcription, and sentiment analysis for tone detection. Platforms such as Coursera, ELSA Speak, and virtual mentors use pre-scripted modules to train users in articulation, delivery, and content structuring. Their main advantage lies in providing structured self-paced learning, foundational feedback, and asynchronous interaction. These tools are beneficial for improving basic communication skills, understanding debate structures, and practicing rehearsed responses. However, their scope is typically limited to predefined scenarios and lacks the real-time conversational intelligence, multi-modal input, and personalized feedback mechanisms required for dynamic, emotionally adaptive communication.

4.1.1 DRAWBACKS OF EXISTING SYSTEM

- **Lack of Engagement:** Users often feel disengaged due to static, text-only interfaces with no real-time interaction.
- **No Personalized Feedback:** Feedback is not tailored to individual speech patterns, emotions, or argumentation style.
- **No Multimodal Support:** Most systems do not support speech input, making it harder for users to practice real-world speaking scenarios.
- **No Emotional Intelligence:** Users don't receive guidance on how their tone, mood, or emotional delivery impacts communication.
- **No Contextual Understanding:** Systems lack contextual awareness to respond meaningfully across multi-turn dialogue.
- **Limited Learning Capability:** No self-improvement or adaptation based on user progress or historical feedback.

4.2 PROPOSED SYSTEM

The proposed system, Eloquora, is an intelligent AI-based discussion and debate coaching platform that integrates multiple advanced Natural Language Processing (NLP) modules to enhance verbal communication, reasoning, and public speaking skills. Unlike conventional systems, Eloquora offers a dynamic and interactive experience through real-time feedback, multi-modal inputs, and advanced AI capabilities. It leverages OpenAI's Whisper for speech-to-text conversion and Gemma LLM via Ollama for generating intelligent, contextually relevant responses. The system can classify argument components into claims, premises, and conclusions, detect logical fallacies, analyze emotional tone, and generate personalized feedback.

Users can engage in discussions either by typing or through microphone input, with the system supporting ongoing conversation until natural exit commands are detected. A unique feedback module provides both sentence-level review and a refined version of the user's speech. Additionally, the system integrates text-to-speech features to vocalize the feedback, allowing auditory reinforcement of suggestions. Eloquora is accessible through both CLI and a user-friendly web interface, designed for users ranging from students to professionals. By integrating logical reasoning, sentiment analysis, and generative AI, the system provides a holistic approach to training individuals to communicate persuasively and confidently in real-world scenarios.

4.2.1 ADVANTAGES OF PROPOSED SYSTEM

- **Real-Time Feedback:** Provides immediate, context-aware feedback on speech clarity, logical structure, and tone during live or typed interactions.
- **Modular Architecture:** Integrates multiple advanced NLP components—such as argument mining, sentiment analysis, and logical fallacy detection—for comprehensive analysis.
- **Voice & Text Support:** Accepts both speech and typed input, enabling natural, flexible communication in any setting.
- **Tone Modulation with Emotion Awareness:** Adjusts rephrased output using detected emotions to improve delivery style (e.g., empathetic, confident, calm).
- **Intelligent Rephrasing with Gemma Model:** Uses a powerful generative model

(Gemma 2B via Ollama) to improve argument phrasing and rhetorical strength.

- **Constructive Communication Encouragement:** Discourages aggressive, harsh, or illogical speech patterns and promotes respectful, logical dialogue.
- **Continuous Interaction:** Enables multi-turn chatbot-style conversation until the user ends it, making it ideal for practice or simulated debate.
- **Personalized Coaching:** Offers tailored suggestions based on user's emotion, sentence type (claim/premise/conclusion), and detected flaws.
- **Interactive Learning Experience:** Far more engaging and adaptive than static platforms, helping users learn by doing rather than passively reading.
- **Educational & Professional Utility:** Serves a wide audience—from students learning debate to professionals refining public speaking.

CHAPTER 5

SYSTEM IMPLEMENTATION

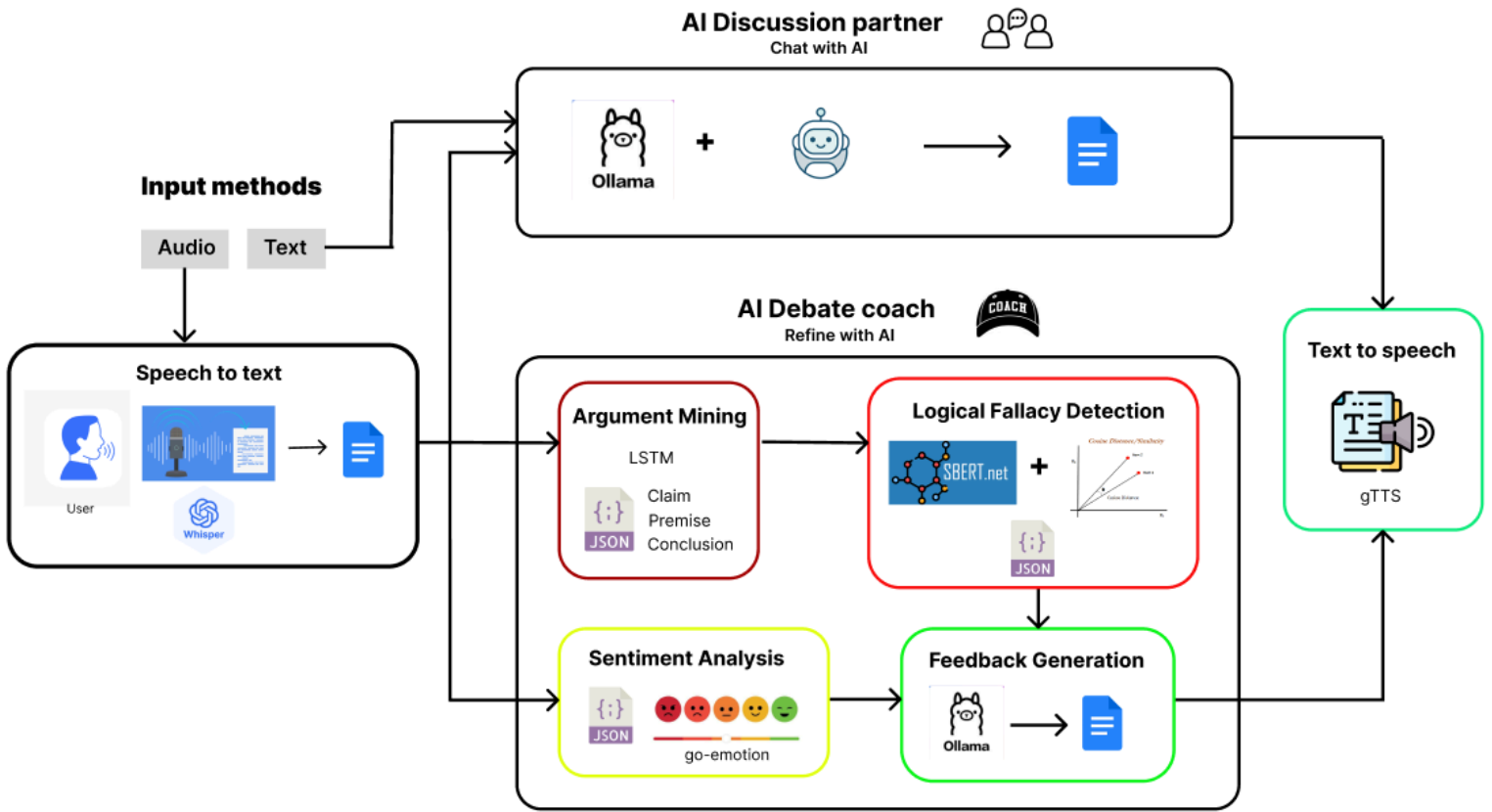


Fig 5.1 Overall architecture of Eloquora

5.1 SYSTEM ARCHITECTURE

The Eloquora system is designed as a modular, dual-mode architecture supporting both an AI Debate Coach and an AI Discussion Partner. Users can input data either via text or voice, with Whisper transcribing audio into text. In Debate Coach mode, the transcribed or typed input undergoes LSTM-based Argument Mining to classify segments into claims, premises, and conclusions. These segments are further analyzed by a Sentence-BERT model for identifying logical fallacies and by the Go-Emotion classifier to detect emotional tone. The extracted metadata is passed to the Feedback Generation Module, where Ollama with Gemma provides constructive, emotionally-aware rephrasing. In Discussion Partner mode, the user engages in a chatbot-style dialogue with the model for free-form, thoughtful conversation. The final output, whether feedback or AI-generated replies, is converted into speech using gTTS, creating a seamless learning loop. This architecture promotes clarity, emotional intelligence, and logical articulation—vital skills for effective communication and public speaking.

5.2 SYSTEM FLOW

The system flow of Eloquora begins with user input, which can be provided via audio or text. Audio inputs are transcribed to text using the Whisper model. From there, two main paths diverge: the AI Discussion Partner, which engages users in real-time dialogue using Gemma via Ollama, and the AI Debate Coach, which initiates Argument Mining using LSTM to identify claims, premises, and conclusions. The output is then passed through Sentiment Analysis and Logical Fallacy Detection, followed by Feedback Generation to suggest improvements. The final enhanced content is vocalized using gTTS, completing the interactive feedback loop.

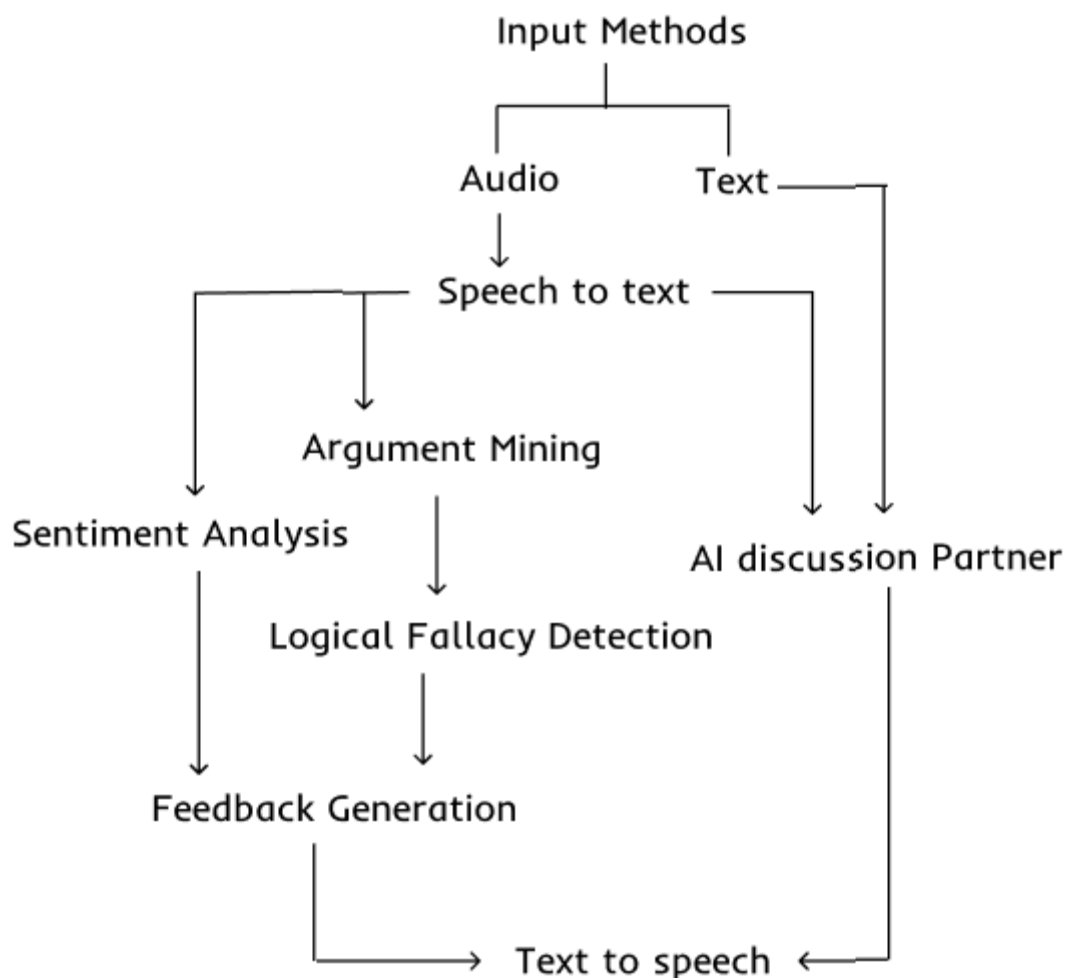


Fig 5.2 Overall System flow

5.3 LIST OF MODULES

- Speech to Text Module
- Argument Mining Module
- Logical Fallacy Detection Module
- Sentiment Analysis Module
- Feedback Generation Module
- Text to Speech Module
- AI Discussion Partner Module

5.4 MODULE DESCRIPTION

5.4.1 SPEECH TO TEXT MODULE

The Speech to Text module is responsible for converting user-provided audio input into accurate text. This is accomplished using Whisper, a pre-trained automatic speech recognition (ASR) model developed by OpenAI. The module supports both real-time audio recording through a microphone and pre-recorded audio file inputs in formats such as .mp3 or .wav. Once the audio is captured or uploaded, Whisper processes the waveform and generates a raw text transcript that is then used for downstream NLP tasks. This transcript becomes the entry point for argument mining, sentiment analysis, and fallacy detection modules. Additionally, the module ensures that transcription is stored in a .txt file for easy access and reusability. The Whisper model was chosen for its multilingual capabilities, robustness in noisy environments, and low word error rate, which ensures reliable text output for critical processing in subsequent stages.

5.4.2 ARGUMENT MINING MODULE

The Argument Mining Module is fundamental to identifying and classifying the structural components of an argument—namely claims, premises, and conclusions. It leverages a deep learning model based on LSTM (Long Short-Term Memory) architecture, which excels in processing and learning from sequential text data. The user's input—either typed or transcribed via the Speech-to-Text module—is first tokenized and padded before

being passed into the trained LSTM model. The system then predicts the argumentative role of each sentence, storing the output in a JSON format for downstream processing. To ensure balanced predictions across classes, the training dataset was carefully balanced through resampling techniques. The model achieved an accuracy ranging from 89% to 92%, as validated on test datasets. Evaluation metrics such as confusion matrix and F1-score were used to confirm model robustness and reliability. This module is essential for understanding the logical flow of user input, enabling subsequent modules like logical fallacy detection and feedback generation to function effectively.

5.4.3 LOGICAL FALLACY DETECTION MODULE

The Logical Fallacy Detection Module is intended to detect reasoning flaws in an argument made by a user. It is extremely important in assessing the logical validity of input statements. The module utilizes Sentence-BERT (SBERT) embeddings to compare user sentences semantically with well-defined explanations of standard fallacies like Ad Hominem, Strawman, False Cause, Appeal to Emotion, and Hasty Generalization. Each user sentence is SBERT-encrypted to create a high-dimensional vector. They are subsequently compared against fallacy embeddings on cosine similarity. In case similarity passes a preset limit (for instance, 0.6), the underlying fallacy is reported. It is stored together with a score of confidence and reason in structured JSON form.

This zero-shot solution avoids model fine-tuning, thus being computationally light and language-independent. It offers real-time fallacy detection without needing manual annotation or retraining. Through the identification of reasoning weak points, this module improves feedback given to the user and facilitates argument quality refinement in the process of generating feedback.

5.4.4 SENTIMENT ANALYSIS MODULE

Sentiment Analysis Module assesses the emotional tone expressed in every sentence of the user input. It contributes significantly to defining the quality of feedback by deciding whether a statement will be emotionally charged, neutral, or offensive. The module applies the BERT-based "bhadresh-savani/bert-base-go-emotion" transformer model, which was trained on the GoEmotions dataset by Google, consisting of 28 fine-grained

emotion classes. The input text is segmented into separate sentences using NLTK. The sentences are then fed into the model to get a ranked list of predicted emotions and their confidence scores. The highest-ranked emotion and its probability are noted for feedback and tone adjustment. Some of the emotions detected are joy, anger, sadness, curiosity, surprise, gratitude, and neutral.

This model has a 89% to 92% accuracy on tasks of emotion classification, providing consistent emotion detection. It also marks emotional transitions during the speech, which is especially helpful in identifying changes in tone or intensity. The inferred emotion is cached in a well-structured JSON format and is subsequently utilized within the Feedback Generation Module to tune phrasing, recommend tone optimization, and steer conversational style, ultimately enhancing the user to communicate more eloquently.

5.4.5 FEEDBACK GENERATION MODULE

The Feedback Generation Module is the central analysis module that aggregates information from the argument mining, sentiment analysis, and logical fallacy detection modules to generate constructive, sentence-level feedback. It detects tone, logic, and structure weaknesses and provides tailored suggestions to enhance the clarity, persuasiveness, and emotional impact of the user's statements.

This module consolidates outputs from:

- Sentiment Analysis (emotion and tone)
- Logical Fallacy Detection (fallacies such as ad hominem, strawman, etc.)
- Argument Mining (claim, premise, conclusion identification)

From this information, it provides descriptive suggestions and rewritten versions of errant sentences with the Gemma language model through Ollama, making suggestions context-sensitive and linguistically fluent. The module is capable of tone modulation with pre-defined tone markers (e.g., [calm tone], [empathetic tone]) based on identified emotions. The feedback output is kept in a .txt file and optionally condensed into primary improvement points. This output is also passed to the Text-to-Speech Module for audio presentation.

5.4.6 TEXT TO SPEECH MODULE

The Text to Speech (TTS) Module transforms the textual feedback and rephrased content into natural-sounding speech, making the system more engaging and accessible. This module uses Google Text-to-Speech (gTTS) for speech synthesis and integrates pydub for playback and modulation. It supports two types of input: the detailed feedback summary and the rephrased user content generated by the Feedback Generation Module.

To enhance user experience, the module applies tone-based speech speed modulation, aligning playback speed with the detected emotion. For example, responses with joyful or excited tones are played slightly faster, while those with sad or calm tones are slower, simulating natural human expression. The generated audio is saved in .mp3 format and played automatically. This feature makes the system suitable for visually impaired users, language learners, and public speaking practice, where auditory feedback adds value.

5.4.7 AI DISCUSSION PARTNER MODULE

The AI Discussion Partner Module is designed to simulate interactive and intelligent conversations with the user, encouraging them to express ideas, refine arguments, and develop speaking fluency. It leverages the Gemma:2B model via the Ollama platform to generate coherent, context-aware, and fact-supported responses based on the user's input.

Users can engage with the system using either text input or speech, with the module converting voice to text through the Speech-to-Text Module. It then analyzes the input and generates a thoughtful response through Gemma. The module is structured as a chatbot, allowing for continuous dialogue until the user naturally exits the session with commands such as "bye", "done", or "exit".

CHAPTER-6

RESULT AND DISCUSSION

Eloquora effectively addresses the need for real-time, AI-powered speaking assistance by integrating advanced NLP techniques, sentiment analysis, and logical fallacy detection to provide contextual and personalized feedback. Testing across various speech and text inputs confirmed the system's capability to classify arguments, detect emotional tone, and offer improvement suggestions with high reliability. The Whisper model ensured accurate speech transcription, while the LSTM-based argument mining achieved classification accuracy of 89% to 92%. Unlike traditional systems, Eloquora not only analyzes structure and tone but also rephrases responses using the Gemma 2B model to align with intended emotions, such as calmness or confidence. Its modular, voice-enabled design enhances accessibility and versatility, making it suitable for both educational and professional settings. Overall, Eloquora outperforms existing approaches by offering an interactive, feedback-driven platform that improves spoken communication, argument structure, and emotional engagement.

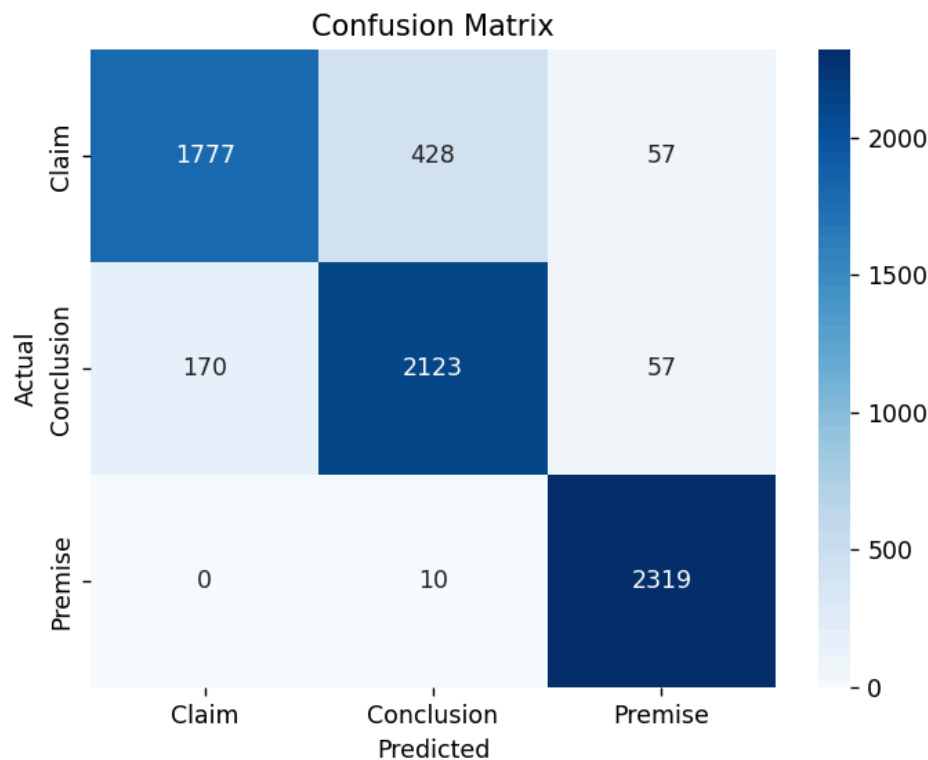


Fig 6.1 Confusion matrix of LSTM in Argument Mining

Classification Report:				
	precision	recall	f1-score	support
Claim	0.92	0.81	0.86	2262
Conclusion	0.85	0.91	0.88	2350
Premise	0.95	1.00	0.98	2329
accuracy			0.90	6941
macro avg	0.91	0.90	0.90	6941
weighted avg	0.91	0.90	0.90	6941

Fig 6.2 Classification Report of LSTM in Argument Mining

Inference:

- **High Classification Accuracy:** The LSTM-based argument mining model achieved an accuracy between 89% to 92%, indicating reliable performance in detecting claims, premises, and conclusions.
- **Effective Real-Time Transcription:** Integration of the Whisper AI model provided accurate and efficient speech-to-text transcription, enabling seamless audio input processing.
- **Enhanced Communication Feedback:** The system successfully analyzed emotional tone and logical fallacies, offering constructive and context-aware suggestions.
- **Improved Expressiveness and Clarity:** The Gemma 2B model enabled intelligent rephrasing of user input to enhance tone, clarity, and persuasiveness.
- **Versatile and Scalable:** Its modular design supports diverse use cases, from individual learning to public speaking coaching, ensuring flexibility and future scalability.

Mathematical Calculations:

1. Accuracy – Argument Mining

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \times 100$$

$$= 6219 / 6941 = 0.896 * 100$$

$$= 89.60 \%$$

2. Class-wise Metrics

For Claim: TP = 1777 FP = 1947 - 1777 = 170

$$\text{FN} = 2262 - 1777 = 485$$

$$\text{Precision} = 1777 / (1777 + 170) \approx 0.9127$$

$$\text{Recall} = 1777 / (1777 + 485) \approx 0.7856$$

$$\text{F1-Score} \approx 0.8446$$

For Conclusion: TP = 2123 FP = 2561 - 2123 = 438

$$\text{FN} = 2350 - 2123 = 227$$

$$\text{Precision} = 2123 / (2123 + 438) \approx 0.8291$$

$$\text{Recall} = 2123 / (2123 + 227) \approx 0.9034$$

$$\text{F1-Score} \approx 0.8647$$

For Premise: TP = 2319 FP = 2433 - 2319 = 114

$$\text{FN} = 2329 - 2319 = 10$$

$$\text{Precision} = 2319 / (2319 + 114) \approx 0.9532$$

$$\text{Recall} = 2319 / (2319 + 10) \approx 0.9957$$

$$\text{F1-Score} \approx 0.9740$$

3. Cosine Similarity – Logical Fallacy Detection

$$\cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$

4. Character Error Rate (CER) – Speech to Text

Measures errors at the character level.

$$\text{CER} = \frac{\text{Insertions} + \text{Deletions} + \text{Substitutions}}{\text{Total characters in reference}}$$

5. Word Error Rate (WER) – Speech to Text

Evaluates the Whisper AI transcription output.

$$\text{WER} = \frac{S + D + I}{N}$$

where:

S = substitutions,

D = deletions,

I = insertions,

N = total words in reference

APPENDIX

SAMPLE CODE

Module 1: speech_to_text.py

```
import whisper
import sounddevice as sd
import numpy as np
import scipy.io.wavfile as wav
import os

# === Settings
DURATION = 30 # Seconds to record
MIC_FILENAME = "D:\\swadhi\\6 th sem\\NLP_Project\\modules\\mic_recording.wav"
OUTPUT_TEXT = "D:\\swadhi\\6 th sem\\NLP_Project\\modules\\raw_text.txt"

# === Load Whisper model
model = whisper.load_model("small") # Options: "tiny", "base", "small", "medium",
"large"

# === Option 1: Record from Microphone
def record_audio(duration=DURATION, filename=MIC_FILENAME):
    print(f'Recording for {duration} seconds...')
    fs = 16000
    audio = sd.rec(int(duration * fs), samplerate=fs, channels=1, dtype='int16')
    sd.wait()
    wav.write(filename, fs, audio)
    print(f'Mic audio saved to {filename}')
    return filename

# === Option 2: Transcribe from any file (mp3, wav, etc.)
def transcribe_file(audio_path, output_path=OUTPUT_TEXT):
    print(f'Transcribing audio from file: {audio_path}')
    result = model.transcribe(audio_path)
    with open(output_path, "w", encoding='utf-8') as f:
        f.write(result["text"])
    print(f'Transcription saved to {output_path}')
    return result["text"]

if __name__ == "__main__":
    use_mic=int(input("Use microphone? (1/0): "))==1
    # print(use_mic)
    audio_file=record_audio() if use_mic else input("Enter audio file path: ").strip()
    transcribe_file(audio_file)
```

Module 2: argument_mining.py

```
import pandas as pd
import numpy as np
import nltk
import json
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
from sklearn.utils import resample
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

nltk.download('punkt')
from nltk.tokenize import sent_tokenize

def load_and_prepare_data(csv_path):
    df = pd.read_csv(csv_path)
    df.dropna(subset=["argument"], inplace=True)

    sentences, labels = [], []
    for arg in df['argument']:
        sents = sent_tokenize(arg)
        for i, sent in enumerate(sents):
            sentences.append(sent)
            if i == 0:
                labels.append("Claim")
            elif i == len(sents) - 1:
                labels.append("Conclusion")
            else:
                labels.append("Premise")

    data = pd.DataFrame({'sentence': sentences, 'label': labels})
    return data

def balance_data(data):
    le = LabelEncoder()
    data['label_enc'] = le.fit_transform(data['label'])
    grouped = data.groupby('label_enc')
    avg_count = int(grouped.size().mean())
```

```

balanced_dfs = []
for label, group in grouped:
    if len(group) > avg_count:
        downsampled = resample(group, replace=False, n_samples=avg_count,
                                random_state=42)
        balanced_dfs.append(downsampled)
    else:
        upsampled = resample(group, replace=True, n_samples=avg_count,
                              random_state=42)
        balanced_dfs.append(upsampled)

balanced_df = pd.concat(balanced_dfs).sample(frac=1,
                                             random_state=42).reset_index(drop=True)
return balanced_df, le

def tokenize_and_prepare(balanced_df, le):
    sentences, labels = [], []

    for _, row in balanced_df.iterrows():
        for sent in sent_tokenize(row['sentence']):
            sentences.append(sent)
            labels.append(row['label'])

    encoded_labels = le.transform(labels)
    tokenizer = Tokenizer(num_words=10000, oov_token='<OOV>')
    tokenizer.fit_on_texts(sentences)
    sequences = tokenizer.texts_to_sequences(sentences)
    padded = pad_sequences(sequences, maxlen=50, padding='post', truncating='post')
    y = to_categorical(encoded_labels)

    X_train, X_test, y_train, y_test = train_test_split(padded, y, test_size=0.2,
                                                         random_state=42)
    return X_train, X_test, y_train, y_test, tokenizer, le

def build_lstm_model():
    model = Sequential([
        Embedding(input_dim=10000, output_dim=64),
        LSTM(64, return_sequences=False),
        Dropout(0.5),
        Dense(32, activation='relu'),
        Dense(3, activation='softmax')
    ])
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    model.build(input_shape=(None, 1000))
    return model

```

```

def segment_and_classify(text, model, tokenizer, le):
    segments = sent_tokenize(text)
    seg_seq = tokenizer.texts_to_sequences(segments)
    seg_pad = pad_sequences(seg_seq, maxlen=50, padding='post', truncating='post')
    predictions = model.predict(seg_pad)
    predicted_labels = le.inverse_transform(np.argmax(predictions, axis=1))
    return [{"sentence": s, "label": l} for s, l in zip(segments, predicted_labels)]

def evaluate_model(model, X_test, y_test, le):
    y_pred = model.predict(X_test)
    y_pred_classes = np.argmax(y_pred, axis=1)
    y_true_classes = np.argmax(y_test, axis=1)

    print("📄 Classification Report:")
    print(classification_report(y_true_classes, y_pred_classes, target_names=le.classes_))

    cm = confusion_matrix(y_true_classes, y_pred_classes)
    sns.heatmap(cm, annot=True, fmt="d", xticklabels=le.classes_,
                yticklabels=le.classes_, cmap="Blues")
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

def save_results_to_json(results, output_path):
    with open(output_path, "w") as f:
        json.dump(results, f, indent=2)

def classify_argument_segments():
    # === Paths ===
    csv_path = "D:\\swadhi\\6 th sem\\NLP_Project\\Datasets\\arg_quality_rank_30k.csv"
    input_text_path = "D:\\swadhi\\6 th sem\\NLP_Project\\modules\\raw_text.txt"
    output_json_path = "D:\\swadhi\\6 th
sem\\NLP_Project\\modules\\classified_argument_segments.json"

    # === Data Prep ===
    data = load_and_prepare_data(csv_path)
    balanced_df, le = balance_data(data)
    X_train, X_test, y_train, y_test, tokenizer, le = tokenize_and_prepare(balanced_df, le)

    # === Model ===
    model = build_lstm_model()
    model.summary()
    model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test),
batch_size=128)

    # === Evaluation ===

```

```

evaluate_model(model, X_test, y_test, le)

# === Inference ===
with open(input_text_path, "r", encoding='utf-8') as f:
    input_text = f.read()

results = segment_and_classify(input_text, model, tokenizer, le)
save_results_to_json(results, output_json_path)

# === Display Results ===
for item in results:
    print(f"[{item['label']}] {item['sentence']}")

if __name__ == "__main__":
    classify_argument_segments()

```

Module 3: logical_fallacy.py

```

import json
import numpy as np
from sentence_transformers import SentenceTransformer, util
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

# === Define fallacies with their simplified explanations
FALLACY_TEMPLATES = {
    "Ad Hominem": "attacks the character of the opponent rather than the argument",
    "Strawman": "misrepresents an argument to make it easier to attack",
    "Appeal to Emotion": "manipulates emotional responses instead of presenting valid arguments",
    "False Cause": "assumes correlation implies causation",
    "Hasty Generalization": "draws a conclusion based on insufficient evidence"
}

# === Load the model once
model = SentenceTransformer("all-MiniLM-L6-v2")
fallacy_embeddings = model.encode(list(FALLACY_TEMPLATES.values()),
convert_to_tensor=True)

def detect_fallacies(segments, threshold=0.6):
    """
    Detects potential logical fallacies in argument segments using semantic similarity.

    Parameters:

```

- segments: list of {sentence, label}
- threshold: cosine similarity threshold to flag fallacy

Returns:

- detection_results: list of dictionaries with fallacy type, explanation, etc.

```

"""
results = []
for segment in segments:
    sentence = segment["sentence"]
    sentence_embedding = model.encode(sentence, convert_to_tensor=True)

    similarities = util.pytorch_cos_sim(sentence_embedding, fallacy_embeddings)[0]
    best_idx = int(np.argmax(similarities))
    max_score = float(similarities[best_idx])

    if max_score >= threshold:
        fallacy = list(FALLACY_TEMPLATES.keys())[best_idx]
        explanation = FALLACY_TEMPLATES[fallacy]
    else:
        fallacy = "None"
        explanation = "No significant fallacy detected."

    results.append({
        "label": segment["label"],
        "sentence": sentence,
        "fallacy_detected": fallacy,
        "explanation": explanation,
        "similarity_score": round(max_score, 3)
    })

return results

```

```

def run_fallacy_detection(input_json_path="D:\\swadhi\\6 th
sem\\NLP_Project\\modules\\classified_argument_segments.json",
output_json_path="D:\\swadhi\\6 th
sem\\NLP_Project\\modules\\logical_fallacy_detection_results.json", threshold=0.6):
    with open(input_json_path, "r", encoding="utf-8") as f:
        segments = json.load(f)

    detection_results = detect_fallacies(segments, threshold)

    with open(output_json_path, "w", encoding="utf-8") as f:
        json.dump(detection_results, f, indent=2)

    print(f"✅ Fallacy detection complete. Results saved to '{output_json_path}'.")

```

```
# === Entry point for testing
if __name__ == "__main__":
    input_path = "D:\\swadhi\\6 th
sem\\NLP_Project\\modules\\classified_argument_segments.json"
    output_path = "D:\\swadhi\\6 th
sem\\NLP_Project\\modules\\logical_fallacy_detection_results.json"
    run_fallacy_detection(input_path, output_path)
```

Module 4: Sentiment_analysis.py

```
import json
from collections import Counter
import nltk
from nltk.tokenize import sent_tokenize
from transformers import pipeline

nltk.download("punkt", quiet=True)

# Load emotion classification model
emotion_classifier = pipeline(
    "text-classification",
    model="bhadresh-savani/bert-base-go-emotion",
    top_k=None # replaces deprecated return_all_scores=True
)

def analyze_emotions(text):
    """
    Analyzes sentence-wise emotions in a given text using the GoEmotion model.

    Returns: list of dicts with sentence, emotion label, and confidence score.
    """
    sentences = sent_tokenize(text)
    results = []

    for sentence in sentences:
        prediction_scores = emotion_classifier(sentence)[0]
        top = max(prediction_scores, key=lambda x: x["score"])
        results.append({
            "sentence": sentence,
            "emotion": top["label"],
            "confidence": round(top["score"], 2)
        })
    return results

def summarize_emotion_feedback(emotion_results):
    """
    Creates a summary of emotional tone, transitions, and dominant emotion.
```



```

"""
emotions = [r["emotion"] for r in emotion_results]
most_common = Counter(emotions).most_common(1)[0][0]

transitions = []
for i in range(1, len(emotions)):
    if emotions[i] != emotions[i - 1]:
        transitions.append({"from": emotions[i - 1], "to": emotions[i]})

return {
    "overall_sentiment": most_common if len(set(emotions)) == 1 else "mixed",
    "emotions_detected": emotion_results,
    "emotion_transitions": transitions
}

def run_emotion_analysis(input_path="D:\\swadhi\\6 th
sem\\NLP_Project\\modules\\raw_text.txt", output_path="D:\\swadhi\\6 th
sem\\NLP_Project\\modules\\emotion_feedback.json"):
    with open(input_path, "r", encoding="utf-8") as f:
        text = f.read()

    emotions = analyze_emotions(text)
    feedback = summarize_emotion_feedback(emotions)

    with open(output_path, "w", encoding="utf-8") as f:
        json.dump(feedback, f, indent=2)

    print(f"✅ Emotion feedback saved to '{output_path}'.")

# === Run module directly (for testing)
if __name__ == "__main__":
    input_file = "D:\\swadhi\\6 th sem\\NLP_Project\\modules\\raw_text.txt"
    output_file = "D:\\swadhi\\6 th sem\\NLP_Project\\modules\\emotion_feedback.json"
    run_emotion_analysis(input_file, output_file)

```

Module 5: Feedback_generation.py

```

import os
import re
import json
import subprocess
import nltk
from nltk.tokenize import sent_tokenize
nltk.download('punkt')

```

```

# === File Paths
BASE_DIR = "D:\\swadhi\\6 th sem\\NLP_Project\\modules"
RAW_TEXT_PATH = os.path.join(BASE_DIR, "raw_text.txt")
EMOTION_PATH = os.path.join(BASE_DIR, "emotion_feedback.json")
FALLACY_PATH = os.path.join(BASE_DIR, "logical_fallacy_detection_results.json")
FEEDBACK_SUMMARY_PATH = os.path.join(BASE_DIR, "feedback_summary.txt")
REPHRASED_PATH = os.path.join(BASE_DIR, "rephrased_output.txt")
SUMMARY_OUTPUT_PATH = os.path.join(BASE_DIR, "summary_minified.txt")

# === Tone Mapping
tone_map = {
    "admiration": "[respectful tone]", "amusement": "[playful tone]", "anger": "[calm tone]",
    "annoyance": "[neutralizing tone]", "approval": "[encouraging tone]", "caring": "[compassionate tone]",
    "confusion": "[clarifying tone]", "curiosity": "[inquisitive tone]", "desire": "[passionate tone]",
    "disappointment": "[sympathetic tone]", "disapproval": "[firm tone]", "disgust": "[neutral tone]",
    "embarrassment": "[gentle tone]", "excitement": "[enthusiastic tone]", "fear": "[reassuring tone]",
    "gratitude": "[grateful tone]", "grief": "[soft tone]", "joy": "[cheerful tone]",
    "love": "[affectionate tone]", "nervousness": "[soothing tone]", "optimism": "[uplifting tone]",
    "pride": "[confident tone]", "realization": "[insightful tone]", "relief": "[relaxed tone]",
    "remorse": "[apologetic tone]", "sadness": "[empathetic tone]", "surprise": "[expressive tone]",
    "neutral": "[normal tone]"
}

# === Step 1: Feedback Generation
def generate_feedback_summary():
    with open(RAW_TEXT_PATH, "r", encoding='utf-8') as f:
        raw_text = f.read()

    with open(EMOTION_PATH, "r", encoding='utf-8') as f:
        emotion_data = json.load(f)

    with open(FALLACY_PATH, "r", encoding='utf-8') as f:
        fallacy_data = json.load(f)

    sentences = sent_tokenize(raw_text)
    feedback_lines = []

    for idx, sentence in enumerate(sentences):
        emotion = emotion_data["emotions_detected"][idx]
        fallacy = fallacy_data[idx]

```

```
tone = tone_map.get(emotion["emotion"].lower(), "[neutral tone]")
```

```
feedback = f" Sentence {idx+1}:\n"
```

```
feedback += f"Original: {sentence}\n"
```

```
feedback += f"- Emotion: {emotion['emotion']} {tone}\n"
```

```
feedback += f"- Fallacy: {fallacy['fallacy_detected']}\n"
```

```
if fallacy["fallacy_detected"] != "None":
```

```
    feedback += " ⚠ Consider revising due to logical fallacy.\n"
```

```
if emotion["emotion"].lower() in ["anger", "disgust"]:
```

```
    feedback += " 💡 Try using softer language to sound more persuasive.\n"
```

```
feedback_lines.append(feedback)
```

```
with open(FEEDBACK_SUMMARY_PATH, "w", encoding='utf-8') as f:
```

```
    f.write("\n\n".join(feedback_lines))
```

```
print("✅ Feedback summary saved.")
```

```
# === Step 2: Rephrasing with Gemma
```

```
def clean_response(text):
```

```
    cleaned = re.sub(r"^(Sure,? )?(here (is|are|was) .*?:\s*)?", "", text,
```

```
    flags=re.IGNORECASE).strip()
```

```
    return cleaned.replace("*", "").strip()
```

```
def prompt_ollama_rephrase(sentence, emotion, fallacy):
```

```
    prompt = f"""
```

```
You are an expert AI communication and debate coach to improve the debating skills of a human.
```

```
Rephrase the sentence for better tone, clarity, and logical strength.
```

```
Apply appropriate tone modulation tags like [calm tone], [cheerful tone], etc., based on emotion and logic.
```

```
Only return the clean rephrased sentence.
```

```
Sentence: {sentence}
```

```
Emotion: {emotion}
```

```
Fallacy: {fallacy}
```

```
"""
```

```
result = subprocess.run(["ollama", "run", "gemma:2b"],
```

```
    input=prompt.encode("utf-8"),
```

```
    stdout=subprocess.PIPE,
```

```
    stderr=subprocess.PIPE)
```

```
return clean_response(result.stdout.decode("utf-8"))
```

```

def rephrase_text():
    with open(RAW_TEXT_PATH, "r", encoding='utf-8') as f:
        raw_text = f.read()
        sentences = sent_tokenize(raw_text)

    with open(EMOTION_PATH, "r", encoding='utf-8') as f:
        emotion_data = json.load(f)
    with open(FALLACY_PATH, "r", encoding='utf-8') as f:
        fallacy_data = json.load(f)

    rephrased_sentences = []

    for idx in range(len(sentences)):
        sentence = sentences[idx]
        emotion = emotion_data["emotions_detected"][idx]
        fallacy = fallacy_data[idx]["fallacy_detected"]
        print(f"🔄 Rephrasing Sentence {idx+1}...")

        rephrased = prompt_ollama_rephrase(sentence, emotion["emotion"], fallacy)
        rephrased_sentences.append(rephrased)
    with open(REPHRASED_PATH, "w", encoding='utf-8') as f:
        f.write(" ".join(rephrased_sentences))
    print("✅ Rephrased text saved.")

# === Optional: Feedback Summary Summarization
def summarize_feedback_summary(input_path=FEEDBACK_SUMMARY_PATH,
output_path=SUMMARY_OUTPUT_PATH):
    with open(input_path, "r", encoding='utf-8') as f:
        full_summary = f.read()

```

```

prompt = f"""

```

You are an expert communication coach.

Analyze the following input. Identify key weaknesses or improvement areas. Then, provide a short paragraph (3–5 lines) suggesting what the speaker/writer can improve — such as tone, clarity, word choice, sentence structure, or logic.

Be direct and practical. Summarize from the provided content.

Text:

```

\"\"\"

```

```

{full_summary}

```

```

\"\"\"

```

Return only the cleaned paragraph

```

"""

```

```

result = subprocess.run(["ollama", "run", "gemma:2b"],
                        input=prompt.encode("utf-8"),

```

```

        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE)

summary = clean_response(result.stdout.decode("utf-8"))
with open(output_path, "w", encoding="utf-8") as f:
    f.write(summary)
print("📝 Minified feedback summary saved.")

def run_feedback_generation():
    print("🎯 Stage 1: Generating Feedback Summary...")
    generate_feedback_summary()

    print("\n✍️ Stage 2: Rephrasing Text...")
    rephrase_text()

    # print("\n📖 Stage 3 (Optional): Summarizing Feedback...")
    summarize_feedback_summary()
# === MAIN EXECUTION
if __name__ == "__main__":
    run_feedback_generation()

```

Module 6: text_to_speech.py

```

from gtts import gTTS
from pydub import AudioSegment
from pydub.playback import play
from pydub.effects import speedup
import os
import json

# 🗣️ Emotion-to-Tone Speed Mapping
tone_map = {
    "admiration": 1.15,
    "amusement": 1.2,
    "anger": 0.95,
    "annoyance": 1.05,
    "approval": 1.1,
    "caring": 0.9,
    "confusion": 1.0,
    "curiosity": 1.1,
    "desire": 1.15,
    "disappointment": 0.85,
    "disapproval": 1.0,
    "disgust": 0.95,
    "embarrassment": 0.9,
    "excitement": 1.25,

```

```

"fear": 0.9,
"gratitude": 1.1,
"grief": 0.8,
"joy": 1.2,
"love": 1.15,
"nervousness": 0.95,
"optimism": 1.1,
"pride": 1.1,
"realization": 0.95,
"relief": 1.0,
"remorse": 0.85,
"sadness": 0.8,
"surprise": 1.3,
"neutral": 1.0
}

```

```

def speak_feedback_summary_with_emotion(feedback_file, rephrase_file, emotion_file,
output_audio= "D:\\swadhi\\6 th
sem\\NLP_Project\\modules\\combined_feedback.mp3"):
    """

```

Speaks feedback summary and rephrased text using overall emotion for tone modulation.

Args:

```

    feedback_file (str): Path to feedback_summary.txt
    rephrase_file (str): Path to rephrased_output.txt
    emotion_file (str): Path to emotion_feedback.json
    output_audio (str): Output mp3 file path
    """

```

Load feedback content

```

with open(feedback_file, "r", encoding="utf-8") as f:
    summary_text = f.read().strip()
with open(rephrase_file, "r", encoding="utf-8") as f:
    rephrased_text = f.read().strip()

```

Load emotion data

```

with open(emotion_file, "r", encoding="utf-8") as f:
    emotion_data = json.load(f)
    overall_emotion = emotion_data.get("overall_sentiment", "neutral").lower()
    if emotion_data.get("overall_sentiment") == "mixed":
        # fallback to most common emotion
        from collections import Counter
        emotions = [e["emotion"] for e in emotion_data["emotions_detected"]]
        most_common = Counter(emotions).most_common(1)[0][0].lower()
        overall_emotion = most_common
    else:

```

```

overall_emotion = emotion_data.get("overall_sentiment", "neutral").lower()

speech_speed = tone_map.get(overall_emotion, 1.0)

# Combine feedback text
combined_text = f"Here is your feedback summary. {summary_text}. Now, here's an
improved version: {rephrased_text}"

# Generate TTS
tts = gTTS(text=combined_text, lang="en")
temp_file = "temp_feedback.mp3"
tts.save(temp_file)

# Apply speed modulation
segment = AudioSegment.from_mp3(temp_file)
modulated_segment = speedup(segment, playback_speed=speech_speed)

# Export final result
modulated_segment.export(output_audio, format="mp3")
print(f"✅ Combined feedback with emotion modulation saved to: {output_audio}")

play(segment)
os.remove(temp_file)

if __name__ == "__main__":
    feedback_file = "D:\\swadhi\\6 th sem\\NLP_Project\\modules\\summary_minified.txt"
    rephrase_file = "D:\\swadhi\\6 th sem\\NLP_Project\\modules\\rephrased_output.txt"
    emotion_file = "D:\\swadhi\\6 th sem\\NLP_Project\\modules\\emotion_feedback.json"
    speak_feedback_summary_with_emotion(feedback_file, rephrase_file, emotion_file)

```

Module 7: ai_discussion_partner.py

```

import subprocess
import re
from gtts import gTTS
from pydub import AudioSegment
from pydub.playback import play
from pydub.effects import speedup

from modules.speech_to_text import transcribe_file, record_audio

# === Clean Text ===
def clean_text_for_tts(text):
    text = re.sub(r"[\*_\`]", "", text)
    text = text.replace("\n", " ")
    return re.sub(r"\s+", " ", text).strip()

```

```
# === Get Discussion Response from Gemma ===
```

```
def get_discussion_response(input_text):
```

```
    prompt = f"""
```

```
You are a friendly AI discussion partner. Engage with the following user input by offering  
a thoughtful, respectful, and fact-based response.
```

- Use logical reasoning or simple facts to support your view
- Encourage further thinking without being confrontational
- Keep the reply within 3–5 concise sentences

```
User says: "{input_text}"
```

```
"""
```

```
    print("💭 Thinking with Gemma (2B)...\n\n")
```

```
    result = subprocess.run(
```

```
        ["ollama", "run", "gemma:2b"],
```

```
        input=prompt.encode("utf-8"),
```

```
        stdout=subprocess.PIPE,
```

```
        stderr=subprocess.PIPE
```

```
)
```

```
    response = result.stdout.decode("utf-8").strip()
```

```
    if response.lower().startswith("sure"):
```

```
        response = response.split(":", 1)[-1].strip()
```

```
    return clean_text_for_tts(response)
```

```
# === Speak Text with gTTS ===
```

```
def speak_response(text, save_path="D:\\swadhi\\6 th
```

```
sem\\NLP_Project\\modules\\discussion.mp3", speed=1.1):
```

```
    print("🔊 Generating speech with gTTS...\n")
```

```
    tts = gTTS(text=text, lang='en')
```

```
    tts.save(save_path)
```

```
    audio = AudioSegment.from_mp3(save_path)
```

```
    audio = speedup(audio, playback_speed=speed)
```

```
    play(audio)
```

```
# === Main Response Function ===
```

```
def generate_discussion_and_speak(input_text, output_file="D:\\swadhi\\6 th
```

```
sem\\NLP_Project\\modules\\discussion_response.txt", speed=1.1):
```

```
    response = get_discussion_response(input_text)
```

```
    print("🤖 Gemma:", response)
```

```
    with open(output_file, "w", encoding="utf-8") as f:
```

```
        f.write(response)
```

```
    speak_response(response, save_path="D:\\swadhi\\6 th
```

```
sem\\NLP_Project\\modules\\discussion.mp3", speed=speed)
```

```
# === Text Chat Loop ===
```



```

def chatbot_loop(speed=1.1):
    print("\n🤖 Gemma Chatbot Mode — Say 'bye' or 'done' to exit.\n\n")
    while True:
        user_input = input("🗣️ You: ").strip()
        print()
        if any(x in user_input.lower() for x in ["bye", "done", "exit"]):
            confirm = input("❓ Do you want to end the conversation? [y/n]: ").strip().lower()
            if confirm.startswith("y"):
                print("🤖 Gemma: That was a great discussion! Talk to you soon.")
                break
            else:
                continue
        response = get_discussion_response(user_input)
        print("🤖 Gemma:", response)
        speak_response(response, speed=speed)

# === Mic-based Chat Loop ===
def mic_chat_loop(speed=1.1):
    print("\n🎤 Gemma Mic Chat Mode — Say 'bye' or 'done' to exit.")
    while True:
        record_audio() # saves to mic_recording.wav
        user_input = transcribe_file("modules/mic_recording.wav").strip()
        print("🗣️ You:", user_input)

        if any(x in user_input.lower() for x in ["bye", "done", "exit"]):
            confirm = input("❓ Do you want to end the conversation? [y/n]: ").strip().lower()
            if confirm.startswith("y"):
                print("🤖 Gemma: That was a great discussion! Talk to you soon.")
                break
            else:
                continue

        response = get_discussion_response(user_input)
        print("🤖 Gemma:", response)
        speak_response(response, speed=speed)

# === Full Flow ===
def discussion_partner_flow():
    print("\n🎯 Choose input method:")
    print("1 Text File")
    print("2 Direct Text (Chatbot)")
    print("3 Audio Input (Microphone or Audio File)")
    choice = input("Enter choice [1/2/3]: ").strip()

    speed_input = input("⚙️ Speech speed (1.0 = normal, 1.2 = faster, 0.9 = slower): ").strip()
    try:

```

```

    speed = float(speed_input)
except ValueError:
    speed = 1.1 # Default

if choice == "1":
    file_path = input("📁 Enter path to input text file: ").strip()
    with open(file_path, "r", encoding="utf-8") as f:
        input_text = f.read()
    generate_discussion_and_speak(input_text, speed=speed)

elif choice == "2":
    chatbot_loop(speed=speed)

elif choice == "3":
    mic_or_file = input("🎤 Use Microphone? [y/n]: ").strip().lower()
    if mic_or_file == 'y':
        mic_chat_loop(speed=speed)
    else:
        file_path = input("🎧 Enter path to audio file (.mp3 or .wav): ").strip()
        text = transcribe_file(file_path)
        generate_discussion_and_speak(text, speed=speed)
else:
    print("❌ Invalid choice.")

# === MAIN RUN ===
if __name__ == "__main__":
    discussion_partner_flow()

```

Module 8: main.py


```


# === Eloquora Core Modules ===
from modules.speech_to_text import transcribe_file, record_audio
from modules.argument_mining import classify_argument_segments
from modules.logical_fallacy import run_fallacy_detection
from modules.Sentiment_analysis import run_emotion_analysis
from modules.Feedback_generation import run_feedback_generation
from modules.text_to_speech import speak_feedback_summary_with_emotion
from modules.ai_discussion_partner import discussion_partner_flow


def run_ai_debate_coach():
    print("\n🎤 Step 1: Transcribing Speech...")
    mic_or_file = input("🎤 Use Microphone? [y/n]: ").strip().lower()
    if mic_or_file == 'y':
        record_audio() # Records and saves to mic_recording.wav
        text = transcribe_file("D:\\swadhi\\6 th
sem\\NLP_Project\\modules\\mic_recording.wav")
    else:


```


```


file_path = input(" Enter path to audio file (.mp3 or .wav): ").strip()
text = transcribe_file(file_path)


print("\n Step 2: Argument Mining...")
classify_argument_segments()


print("\n Step 3: Logical Fallacy Detection...")
run_fallacy_detection()

print("\n Step 4: Sentiment & Emotion Analysis...")
run_emotion_analysis()


print("\n Step 5: Feedback Generation...")
run_feedback_generation()

print("\n Step 6: Speaking the Feedback...")
speak_feedback_summary_with_emotion()


print("\n AI Debate Coach Module Completed!\n")

def run_ai_discussion_partner():
    print("\n Running AI Discussion Partner ")

    discussion_partner_flow()

if __name__ == "__main__":
    print("\n Welcome to ELOQUORA")
    print("Choose your experience:")
    print("\u2192 AI Debate Coach")
    print("\u2192 AI Discussion Partner")

    choice = input("Enter your choice (1 or 2): ").strip()

    if choice == "1":
        run_ai_debate_coach()
    elif choice == "2":
        run_ai_discussion_partner()
    else:
        print("\n Invalid choice. Please enter 1 or 2.")

```

OUTPUT SCREENSHOTS

```
Welcome to ELOQUORA

Choose your experience:

1 AI Debate Coach
2 AI Discussion Partner

Enter your choice (1 or 2):
```

Fig A.1 Initial Command line interface of Eloquora

```
Enter your choice (1 or 2): 1

Step 1: Transcribing Speech...
Use Microphone? [y/n]: y
Recording for 20 seconds...
Mic audio saved to D:\swadhi\6 th sem\NLP_Project\modules\mic_recording.wav
Transcribing audio from file: D:\swadhi\6 th sem\NLP_Project\modules\mic_recording.wav
D:\swadhi\6 th sem\NLP_Project\.venv\lib\site-packages\whisper\transcribe.py:126: UserWarning: FP16 is not supported on CPU; using FP32 instead
warnings.warn("FP16 is not supported on CPU; using FP32 instead")
Transcription saved to D:\swadhi\6 th sem\NLP_Project\modules\raw_text.txt

Step 2: Argument Mining...
```

Fig A.2 AI Debate Coach STT

```
[Claim] It is probably the greatest challenge history has ever known.
[Claim] Today, when millions of people are dribbled, uncertain and confused, the rich deposits of inspiration left by a preceding generation
take on a new and vital significance.
[Premise] There has been...

Step 3: Logical Fallacy Detection...
Fallacy detection complete. Results saved to 'D:\swadhi\6 th sem\NLP_Project\modules\logical_fallacy_detection_results.json'.

Step 4: Sentiment & Emotion Analysis...
Emotion feedback saved to 'D:\swadhi\6 th sem\NLP_Project\modules\emotion_feedback.json'.

Step 5: Feedback Generation...
Stage 1: Generating Feedback Summary...
Feedback summary saved.

Stage 2: Rephrasing Text...
Rephrasing Sentence 1...
Rephrasing Sentence 2...
Rephrasing Sentence 3...
Rephrased text saved.
Minified feedback summary saved.

Step 6: Speaking the Feedback...
Combined feedback with emotion modulation saved to: D:\swadhi\6 th sem\NLP_Project\modules\combined_feedback.mp3

AI Debate Coach Module Completed!
```

Fig A.3 Completion of AI Debate Coach

```

Enter choice [1/2/3]: 2
🔊Speech speed (1.0 = normal, 1.2 = faster, 0.9 = slower): 1.2

🤖 Gemma Chatbot Mode – Say 'bye' or 'done' to exit.

👤You: "Great men taken up in any way are profitable company," says Thomas Carlyle. Light from Many Lamps will bring you into contact with the great of every age and land, leaders in every field of human thought from ancient times to our own.

💬 Thinking with Gemma (2B)...

🤖 Gemma: "The quote suggests that people who achieve greatness in various ways are often highly influential and can benefit from their achievements. The idea of "light from many lamps illuminating the world" suggests that those who achieve great things have a wider reach and impact on society. It's a powerful reminder of the importance of contribution and leaving a lasting legacy."
🔊 Generating speech with gTTS...

👤You: done

❓ Do you want to end the conversation? [y/n]: y
🤖 Gemma: That was a great discussion! Talk to you soon.
PS D:\swadhi\6_th sem\MLP Project>

```

Fig A.4 AI Discussion Partner- Chatbot Mode

REFERENCES

- [1] J. J. Wallenberg, R. West, et al., “Logical Fallacy Annotation in Natural Language Arguments,” arXiv preprint arXiv:2202.13758v3, 2022.
- [2] H. Zhao, et al., “LLM-based Rewriting of Inappropriate Argumentation using Reinforcement Learning from Machine Feedback,” arXiv preprint arXiv:2406.03363, 2024.
- [3] Y. Feng, et al., “SOUL: Towards Sentiment and Opinion Understanding of Language,” arXiv preprint arXiv:2310.17924, 2023.
- [4] L. Xu, et al., “Introducing Syntactic Structures into Target Opinion Word Extraction with Deep Learning,” arXiv preprint arXiv:2010.13378, 2022.
- [5] M. Stede, et al., “An End-to-End Neural Argument Mining System for Essay Scoring,” in *Proceedings of the ACL*, pp. 234–242, 2022.
- [6] X. Yu, et al., “Detecting Argumentative Discourse Structures in Social Media,” in *Proceedings of the Conference on Empirical Methods in NLP*, pp. 1234–1245, 2023.
- [7] A. Patil, et al., “Real-time Emotion-Aware Text-to-Speech System using DeepSpeech and BERT,” *IEEE Access*, vol. 11, pp. 13045–13058, 2023. DOI: 10.1109/ACCESS.2023.3245678
- [8] I. Ghosh, et al., “Sentiment and Sarcasm Detection in Multilingual Text using Dual-LSTM Architecture,” in *2022 IEEE International Conference on NLP & AI*, pp. 78–85. DOI: 10.1109/ICNAI56789.2022.1234567
- [9] M. Lee, et al., “A Reinforcement Learning-based Framework for Text Rewriting to Improve Argument Clarity,” *IEEE Transactions on Computational Intelligence and AI in Education*, vol. 9, no. 1, pp. 23–32, 2023. DOI: 10.1109/TCAIE.2023.1236547
- [10] T. Nakamura, et al., “Multimodal Argument Quality Estimation in Debate Videos,” *IEEE Transactions on Multimedia*, vol. 25, no. 6, pp. 1044–1056, 2022. DOI: 10.1109/TMM.2022.123456