An Artificial Intelligence-Based Debate Analyser for Argument Mining and Fallacy Detection

By

SIM SAU YANG

TP065596

APU3F2502CS(DA)

A report submitted in partial fulfilment of the requirements for the degree of

B.Sc. (Hons) Computer Science Specialism in Data Analytics

at Asia Pacific University of Technology and Innovation.

Supervised by Mr. Raheem Mafas

2nd Marker: Ms. Mary Ting

2025

DECLARATION OF THESIS CONFIDENTIALITY

Author's full name: Sim Sau Yang

IC No./Passport No.: 030508-01-1219

Thesis/Project title: An Artificial Intelligence-Based Debate Analyser for Argument Mining and Fallacy Detection

---

I declare that this thesis is classified as:

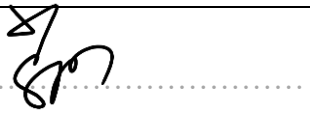
☐ CONFIDENTIAL

☐ RESTRICTED

☑ OPEN ACCESS

I acknowledged that Asia Pacific University of Technology & Innovation (APU) reserves the right as follows:

1. The thesis is the property of Asia Pacific University of Technology & Innovation (APU).
2. The Library of Asia Pacific University of Technology & Innovation (APU) has the right to make copies for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Author's Signature: 

Date: 16 September 2025

Supervisor's Name: **Mr. Raheem Mafas**

Date: 16 September 2025

Signature: RMF

Please fill in **all** the following details for library cataloguing purposes.

First Name: Sau Yang
Middle Name (only if applicable):
Last Name: Sim
Title of the Final Year Project / Dissertation / Thesis: An Artificial Intelligence-Based Debate Analyser for Argument Mining and Fallacy Detection
Abstract: Debate is a powerful channel for practising critical thinking. However, analysing arguments and spotting fallacies at scale is slow and inconsistent. This project develops an AI-based debate analyser that turns raw transcripts into sentence-level feedback. A large subset of IBM Project Debater transcripts is consolidated, cleaned, and segmented with punctuation restoration and discourse-cue splitting. A sampled set of sentences is annotated for four argument types (Claim, Grounds, Warrant or Qualifier, Others) using assisted labelling with manual verification, then expanded to the whole corpus via transfer learning. Multiple classifiers are trained and fairly evaluated, including a TF-IDF Linear SVM, a Text-CNN, and several transformers, which are DeBERTa v3, GPT-2, DeepSeek-R1 with QLoRA, and TinyLLaMA with QLoRA. Hyperparameters are tuned on a development slice of 100 debates, and the best settings are retrained on the complete data. Performance is reported with accuracy, precision, recall, macro-F1, confusion matrices, and ROC curves. The strongest configuration, DeBERTa v3, attains a macro-F1 of 0.91 and 93% accuracy on argument type classification. A lightweight, pretrained fallacy detector is integrated to flag common logical errors, and the final model is deployed in a Streamlit web interface that highlights argument roles and surfaces concise improvement suggestions. By offering fast, structured feedback on reasoning, the system supports Sustainable Development Goal 4 on Quality Education and provides a reproducible baseline for debate-domain argument mining.
A few keywords associated with the work: Argument Mining, Logical Fallacies Detection, Debate Analysis, Fine-Tuning Models
General Subject: Computer Science in Data Analytics
Date of Submission: 17th September 2025

Acknowledgement

Above all, the researcher would like to sincerely thank his supervisor, Mr. Raheem Mafas, a Senior Lecturer, for his unwavering support, invaluable advice, and motivation throughout the Final Year Project. Mr. Raheem was never stingy with his time, going out of his way to hold meetings and offer constructive criticism that enabled the direction and progress of this project to be steered in the right way. Without his patience and experience, the researcher would have faced tremendous difficulty finishing this work.

The researcher would like to thank Ms. Nur Amira Binti Abdul Majid, the Final Year Project manager, for her excellent coordination and precise guidance throughout the procedure. Her highly organised briefings and well-structured resources enabled the researcher to remain on track and effectively progress through each project stage.

The researcher is forever grateful to his family and friends for their unconditional support, encouragement, and patience. They were a source of strength in times of stress, and their motivation and presence kept him going despite difficulties encountered while undertaking this scholarly endeavour.

Finally, the researcher would like to thank himself for being constant, committed, and persistent throughout this project. Despite numerous difficulties and life doubts, he went on with unwavering determination. This project is recognised as a scholastic success and a significant personal success, demonstrating growth and perseverance.

Abstract

Debate is a powerful channel for practising critical thinking. However, analysing arguments and spotting fallacies at scale is slow and inconsistent. This project develops an AI-based debate analyser that turns raw transcripts into sentence-level feedback. A large subset of IBM Project Debater transcripts is consolidated, cleaned, and segmented with punctuation restoration and discourse-cue splitting. A sampled set of sentences is annotated for four argument types (Claim, Grounds, Warrant or Qualifier, Others) using assisted labelling with manual verification, then expanded to the whole corpus via transfer learning. Multiple classifiers are trained and fairly evaluated, including a TF-IDF Linear SVM, a Text-CNN, and several transformers, which are DeBERTa v3, GPT-2, DeepSeek-R1 with QLoRA, and TinyLLaMA with QLoRA. Hyperparameters are tuned on a development slice of 100 debates, and the best settings are retrained on the complete data. Performance is reported with accuracy, precision, recall, macro-F1, confusion matrices, and ROC curves. The strongest configuration, DeBERTa v3, attains a macro-F1 of 0.91 and 93% accuracy on argument type classification. A lightweight, pretrained fallacy detector is integrated to flag common logical errors, and the final model is deployed in a Streamlit web interface that highlights argument roles and surfaces concise improvement suggestions. By offering fast, structured feedback on reasoning, the system supports Sustainable Development Goal 4 on Quality Education and provides a reproducible baseline for debate-domain argument mining.

Keywords: Argument Mining, Logical Fallacies Detection, Debate Analysis, Fine-Tuning Models

Table of Contents

List of Figures

List of Tables

List of Code Snippets

Chapter 1: Introduction

1.1 Introduction

Effective communication is a vital skill that enables individuals to express their ideas clearly, build strong relationships, and achieve their goals in both personal and professional contexts (Aebissa, 2023). Its impact is shown daily in formal debates, public speaking, academic discussions, or casual conversations. However, people often rely on mental shortcuts or heuristics when making decisions, which can unintentionally lead to biased or irrational reasoning (Friedman, 2023). This weakens the quality of discussions and can lead to misunderstandings or the spread of misinformation. Analysing one's or others' arguments to detect logical flaws and improve reasoning requires significant effort and time, making it inefficient when done manually.

With advancements in Artificial Intelligence (AI) and large language models (LLMs), this project proposes an AI-powered system that automatically analyses arguments from debate transcripts. The system identifies argument types, detects logical fallacies, and provides constructive improvement suggestions to help users refine their reasoning. The system fosters improved communication and critical thinking skills by providing immediate feedback on flawed logic and argumentative structure.

This project aligns with Sustainable Development Goal (SDG) 4, Quality Education, and targets inclusive and equitable education and lifelong learning (Goal 4: Quality Education—the Global Goals, 2024). The system promotes independent learning, self-evaluation, and more participatory public debate by helping users understand and refine their arguments. The end product will be an interactive web application that allows users to enter transcripts and receive results on the detection of fallacies, along with constructive feedback to enhance their arguments.

1.2 Problem Background

1. Human Bias in Debates and Speeches Evaluation

People are likely to evaluate arguments based on their own beliefs and not based on objective reasoning, and debate judgments are likely to be biased. Research has shown that people like arguments supporting their existing beliefs, making them less likely to reasonably judge opposing views (Baccini & Hartmann, 2022). This myside bias affects how people construct arguments and evaluate other people's reasoning. Furthermore, individuals tend to overestimate the accuracy of their judgments (overconfidence bias), believe that events can be predicted after they have occurred (hindsight bias), and persist in defending poor arguments because of past investments (sunk cost fallacy) (Berthet, 2021). Such cognitive biases misrepresent the evaluation of arguments, resulting in inaccurate conclusions and unjustified judgments. Studies have also proven that human debate judges are influenced by misinformation, authority, and even appearance (beauty bias), making their judgments subjective and unreliable (Chen et al., 2024). Such biases can compromise the objectivity of debate judgments and mislead people's minds.

2. Manipulative Arguments Mislead People

Individuals are continually exposed to persuasive speech that affects their decisions and opinions without realising it. Rhetoric takes various forms, including propaganda that shapes public information on political and social matters, misinformation that disseminates false or misleading claims, and persuasive advertising strategies that influence consumers' decisions in their favour (Corvaglia, 2024). These arguments may seem reasonable, but they are derived from flawed logic rooted in logical fallacies. Because these fallacies permeate everyday speech, news, and media, people accept them as facts, thereby developing misinformed opinions and making poor decisions (Abd-Eldayem, 2023). Individuals are exposed to biases and deceptive rhetoric that influence their beliefs unrecognised, without a formulated method of detecting and criticising such manipulative arguments. Solving this issue aligns with the aim of Sustainable Development Goal 4 through increased argument analysis competency and logical fallacy identification.

3. Debate and Speech Analysis is Time-Consuming and Inefficient

Manual processing of big text data is lengthy, laborious, and unreliable, making it difficult to efficiently extract useful information (Hacking et al., 2023). Research has established that manual processing of vast amounts of text takes a lot of time and resources, causing delays and unreliability in the analysis. Traditional methods such as rule-based classification and hand coding are time-consuming and generally not feasible for handling large datasets (Ghahreman & Dastjerdi, 2011). Researchers spend countless hours hand-coding arguments and main points in areas such as policy analysis, thereby limiting the potential for scale in such analyses (Gielens & Sowula, 2024). As the frequency of debate and discussion grows, depending on human assessors is becoming increasingly impractical, highlighting the necessity for automated systems to facilitate the assessment and classification of arguments with accuracy and consistency.

1.3 Project Aim

To develop an AI-powered system that analyses debate arguments by identifying argument types, detecting logical fallacies, and providing improvement suggestions.

1.4 Objectives

1. To collect and compile a comprehensive dataset of debate transcripts, including sentence-level argument type labelling and fallacy annotations.
2. To train and fine-tune models to automatically identify argument types and adopt the model to detect logical fallacies from text.
3. To assess and compare the performance of different models using standard evaluation metrics.
4. To develop and deploy an interactive system that analyses input transcripts, detects fallacies, identifies argument types, and provides suggestions for improvement.

1.5 Scope

1.5.1 Dataset

This project uses a collection of debate transcripts from the IBM Project Debater. The dataset consists of thousands of plain text files, each containing a full transcript of a debater's speech.

1.5.2 Deliverables

1. Data Collection and Understanding

The project begins by collecting suitable open-source debate transcripts after comparing datasets from trusted platforms such as IBM Debater, Kaggle, and Hugging Face. Exploratory Data Analysis (EDA) will be conducted to understand the dataset structure, sentence length, and stance.

2. Data Preprocessing

Once the dataset is understood, it will be cleaned and pre-processed. Based on Toulmin's model, simplified manual argument type labelling will be performed on a sample set, followed by transfer learning to expand the labelled dataset. Logical fallacies will be annotated using pre-trained models or expert pipelines from platforms like Hugging Face. Text pre-processing can include structuring the data into sentence-level units, tokenisation, vectorisation, and other text processing steps. The processed data will be explored again to seek more insights.

3. Model Training

After preprocessing, the data will be split into training and testing sets. Selected training models, such as support vector machine (SVM), neural networks, fine-tuned DeBERTa, GPT, LLaMA, and DeepSeek, will be trained to perform argument type classification. These models will learn from the labelled data and be fine-tuned to adapt to the debate domain.

4. Model Evaluation and Optimisation

The trained models will be evaluated using the test dataset to measure performance. Evaluation metrics, such as accuracy, precision, recall, and F1-score, will be used to compare different models. Hyperparameter tuning and optimisation strategies can be applied to improve model accuracy and efficiency.

5. Model Deployment

Once the best-performing model is selected, it will be integrated into a web-based system. This system enables users to input debate or speech transcripts and receive results that include fallacy detection, argument type classification, and AI-generated suggestions for improvement. A simple dashboard will also be developed to visualise the argument structure and detect issues, helping users better understand their arguments and learn how to improve them.

1.5.3 Constraints

1. The system may face challenges due to the variability in speaking styles, transcript formatting, terminology, and debate structures across different topics and speakers.
2. The classification of argument types and fallacies may still be affected by inherent subjectivity in natural language and the complexity of real-world reasoning.
3. The project will be constrained by computational limitations, which will restrict its ability to fine-tune large language models, process massive datasets, and perform hyperparameter tuning.
4. The system will only support English language transcripts, as multilingual argument mining would require separate language models and additional annotated datasets.
5. The limited availability of high-quality, fully annotated debate datasets may affect the diversity, quality, and representativeness of training data used for modelling.
6. Manual annotation and quality checking of argument types will be time-consuming and may introduce inconsistencies or human bias into the dataset.

1.5.4 What will and will not be done as part of the project

In-scope

1. The project will collect and compile English-language debate transcripts from publicly available and reliable sources such as IBM Debater.
2. The project will perform exploratory data analysis (EDA) to understand argument distributions, fallacy types, and common debate patterns in the dataset.
3. The project will clean, preprocess, and structure the raw data by converting debate paragraphs into sentence-level arguments suitable for model training.
4. The project will apply manual argument type labelling and use transfer learning models to expand the labelled dataset for larger-scale training.
5. The project will fine-tune and train various models, including deep learning architectures and transformer-based pretrained language models.
6. The project will evaluate the trained models using standard metrics such as accuracy, precision, recall, and F1-score to measure classification and detection performance.
7. The project will optimise model parameters and apply tuning strategies to improve the accuracy of argument type classification and fallacy detection.
8. The project will develop and deploy a working prototype of a web-based system that allows users to input debate or speech transcripts for automated argument analysis, fallacy detection, and improvement suggestions.

Out-scope

1. The project will not process debate transcripts in languages other than English because of multilingual support and scope limitations.
2. The project will not include expert-level validation or feedback from professional debaters, argumentation theorists, or legal specialists beyond existing publicly available annotations.
3. The system will not be deployed as a commercial-grade or production-level product; it will remain a functional academic prototype for research purposes only.
4. The project will not guarantee complete elimination of bias in the training data or model outputs, as full-scale bias mitigation is outside the project's current scope.

5. The project will not develop a knowledge-based reasoning engine or advanced symbolic logic mapping, focusing only on argument type classification, fallacy detection, and improvement suggestions.
6. The system will not support real-time debate evaluation or live speech analysis; it will be limited to processing uploaded transcripts only.

1.6 Potential Benefit

1.6.1 Tangible Benefit

1. Recommendations for Argument Improvement

The system diagnoses gaps in arguments and proposes constructive revisions. Providing feedback based on recognised fallacies or structural shortcomings enables users to strengthen their arguments, thereby enhancing clarity, logical coherence, and persuasiveness in discussions or presentations.

2. Automated Detection of Fallacies and Biases

The system enables individuals to automatically identify logical fallacies and biased arguments in transcripts of debates and speeches. By identifying faulty argument constructions and manipulative patterns, the system helps users detect misinformation. It minimises the possibility of being swayed by deceptive information in discussions, classrooms, or day-to-day interactions.

3. Increased Efficiency and Time Savings

Manual analysis of arguments is usually time-consuming and needs expert evaluators. This AI-based system automates the task, rendering it more rapid and consistent. Users can get an organised analysis of a debate or speech in seconds, saving time and effort and eliminating the necessity for expert reviewers.

4. Accessible Argumentation Training Tool

The system is a standalone tool designed to improve argumentation skills. It provides users with an inexpensive and extensible method for practising and receiving feedback on their reasoning

ability, thereby circumventing the need for costly coaching or professional debate classes. It therefore enhances better access to organised argumentation learning by students, educators, and the public.

1.6.2 Intangible Benefit

1. Enhancement of Critical Thinking Skills

Engagement with feedback that identifies fallacies and logical contradictions enables users to refine more advanced critical thinking abilities. This system enables individuals to critically analyse arguments, thereby assisting them in developing more consistent and logical stances in both academic and real-life situations.

2. Improvement in Logical Reasoning and Communication

Users can better organise and articulate their ideas as they revise and strengthen arguments through system suggestions. This leads to improved logical thinking and effective debating, writing, and communication.

3. More Confidence in Debating and Public Speaking

Receiving objective and organised feedback enables users to recognise their improvement over time. This, in turn, builds confidence in their ability to construct logical arguments and articulate ideas effectively, thereby improving their comfort level in public speaking or debate situations.

4. Contribution to Sustainable Development Goal (SDG) 4: Quality Education

This project aligns with Sustainable Development Goal 4 by providing a functional resource that facilitates accessible and inclusive education. The model supports independent analysis and critical thinking by enhancing users' reasoning and communication skills through artificial intelligence-driven evaluation, fostering ongoing educational development.

1.7 Target User

1. Debaters

Debaters can utilise the system to strengthen their arguments, identify logical flaws, and anticipate potential counterarguments more effectively. The system is a helpful training tool for debate competitions because it assists in developing more coherent and consistent arguments.

2. Students and Educators

The system can improve students' argumentative writing and critical thinking skills in various areas of study. It gives feedback that helps with essay composition, debate organisation, and class presentation. Teachers can also utilise the system as a teaching tool for public speaking, critical thinking, and communication, thereby providing uniform and constructive feedback to students.

3. Debate Judges

Judges and evaluators in debates can utilise the system as a supporting aid to help them detect logical fallacies and argumentation structure during competitions or practice sessions. Although not substituting for human judgment, the system may assist in rendering more consistent, structured, and less biased conclusions.

4. General Learners and Self-Improvers

Those who wish to enhance their argumentation, communication, or reasoning abilities may utilise the system as a standalone learning tool. It provides readily available and usable feedback to promote independent practice and the acquisition of argumentation proficiency.

1.8 Overview of the FYP Documentation

1.8.1 Chapter 1: Introduction

Chapter 1 begins with a project introduction, providing a general background of argumentation, debate analysis, and how crucial language models are in argument evaluation. It then provides the problem background, noting such issues as human bias in argument evaluation, susceptibility to fallacious arguments, and inefficiency in argument analysis. The project's objectives are stated, with elaborated objectives that outline how the system will evaluate debates and speeches. The scope defines the dataset, system capabilities, and significant limitations. The 'Possible Benefits' section addresses both tangible and intangible benefits, including enhanced argumentation capability and the fostering of critical thinking. Target users are listed, such as debaters, public speakers, students, educators, and anyone with an interest in advancing their reasoning ability. This chapter also presents an overview of the project plan, laying the groundwork for the subsequent sections.

1.8.2 Chapter 2: Literature Review

Chapter 2 comprehensively reviews the research and technical areas relevant to this project. It begins by discussing the core domains of the project’s focus, covering foundational theories and recent developments that support argument analysis using AI. Some domains are broken into sub-sections to explore specific research challenges, existing solutions, and areas for potential improvement. The chapter also includes examining similar systems to understand their approaches and limitations. Then, technical research is carried out to identify the tools, programming language, and environment, as well as the system requirements, best suited for implementing the debate analyser. This chapter ensures the project is guided by established knowledge and informed technology choices.

1.8.3 Chapter 3: Methodology

Chapter 3 talks about the overall research framework and approach chosen for the project by introducing various popular data analytic methodologies, such as CRISP-DM, SEMMA, and KDD, and compares their strengths and weaknesses. From these comparisons, the most suitable methodology for this project is selected and justified, ensuring the process follows a structured path. This chapter guides each stage of the project, which is from problem understanding to deployment.

1.8.4 Chapter 4: Design and Implementation

Chapter 4 focuses on the actual implementation of the debate analyser project, which will begin with the collection of debate transcripts and continue through every important stage of development. It begins with identifying and preparing the dataset, along with exploratory data analysis (EDA), to understand the data structure and identify key patterns. It will also describe data cleaning and processing approaches to make it suitable for modelling tasks. After preparation, various study models are built and trained to classify argument types. This process includes fine-tuning and optimising hyperparameters to get better performance results. In short, this chapter provides a clear picture of how the project is designed, implemented, and continually improved step by step to ensure the end system is effective.

1.8.5 Chapter 5: Results and Discussion

Chapter 5 explains the project outcomes by presenting the results from model training, testing, and evaluation. It discusses the model's performance in classifying argument types using standard metrics to measure accuracy and reliability. Comparisons between different models are conducted to identify their strengths and weaknesses, and to obtain meaningful findings. Additionally, this chapter introduces the deployment of the final model into a user-friendly interface, allowing target users to interact with the debate analyser system. This chapter also reflects the project's results in relation to its objectives and the contributions made to developing a tool for enhancing argument analysis, debate skills, and critical thinking.

1.8.6 Chapter 6: Conclusion

Chapter 6 summarises the whole project and describes the achievements throughout the development of the debate analyser. It involves a critical evaluation to assess the completion of the objectives set at the beginning, such as creating models to classify argument types, detect logical fallacies, and integrate the system into a user interface. It also discusses the project's strengths and contributions to education. Besides, this chapter also points out the limitations and gives recommendations for future improvements. This conclusion means a full stop to this final year project, but also suggests how it can be developed further beyond this study.

1.9 Project Plan

Name of Tasks	Duration	Start Date	End Date	Status
Project Proposal Form	14	21/2/2025	7/3/2025	Completed
Project Specification Form	1	7/3/2025	8/3/2025	Completed
Ethic Form (Fast Track)	1	1/4/2025	2/4/2025	Completed
Acknowledgement	2	1/4/2025	3/4/2025	Completed
Abstract	1	3/4/2025	4/4/2025	Completed
Chapter 1				Completed
1.1 Introduction	1	5/4/2025	6/4/2025	Completed
1.2 Problem Background	2	7/4/2025	9/4/2025	Completed
1.3 Project Aim	1	10/4/2025	11/4/2025	Completed
1.4 Objectives	1	10/4/2025	11/4/2025	Completed
1.5 Scope	1	10/4/2025	11/4/2025	Completed
1.6 Potential Benefits	1	12/4/2025	13/4/2025	Completed
1.7 Target User	1	13/4/2025	14/4/2025	Completed
1.8 Overview of the FPY	1	6/5/2025	7/5/2025	Completed
1.9 Project Plan	1	6/5/2025	7/5/2025	Completed
Chapter 2				Completed
2.1 Introduction	1	15/4/2025	16/4/2025	Completed
2.2 Domain Research	14	16/4/2025	30/4/2025	Completed
2.3 Similar System / Works	3	17/4/2025	20/4/2025	Completed
2.4 Technical Research	1	20/4/2025	21/4/2025	Completed
2.5 Summary	1	20/4/2025	21/4/2025	Completed
Chapter 3				Completed
3.1 Introduction	1	22/4/2025	23/4/2025	Completed
3.2 Methodology	1	22/4/2025	23/4/2025	Completed
3.3 Summary	1	24/4/2025	25/4/2025	Completed
Chapter 4				Completed
4.1 Introduction	1	25/4/2025	26/4/2025	Completed
4.2 Data Collection	4	25/4/2025	29/4/2025	Completed
4.3 Data Understanding and Data Pre-processing	6	30/4/2025	6/5/2025	Completed
4.4 Dataset Preparation	13	12/7/2025	25/7/2025	Completed
4.5 Model Building	25	26/7/2025	20/8/2025	Completed
4.6 Hyperparameter Tuning	25	26/7/2025	20/8/2025	Completed
4.7 Summary	1	20/8/2025	21/8/2025	Completed
Chapter 5				Completed
5.1 Introduction	1	20/8/2025	21/8/2025	Completed
5.2 Dataset Corpus Evaluation	1	20/8/2025	21/8/2025	Completed
5.3 Model Evaluations and Discussions	6	21/8/2025	27/8/2025	Completed
5.4 Model Deployment	11	28/8/2025	8/9/2025	Completed
5.5 Summary	1	8/9/2025	9/9/2025	Completed
Chapter 6				Completed
6.1 Critical Evaluation	6	9/9/2025	15/9/2025	Completed
6.2 Limitations	6	9/9/2025	15/9/2025	Completed
6.3 Recommendations	6	9/9/2025	15/9/2025	Completed
References				Completed
Appendices				Completed
Project Proposal Form	1	15/9/2025	16/9/2025	Completed
Ethics Form	1	15/9/2025	16/9/2025	Completed
Log Sheets	1	15/9/2025	16/9/2025	Completed
Gantt Chart	1	15/9/2025	16/9/2025	Completed
Sample Code Implementation	1	15/9/2025	16/9/2025	Completed
Turnitin Similarity Report	1	15/9/2025	16/9/2025	Completed

Figure 1: Project Plan

Chapter 2: Literature Review

2.1 Introduction

This chapter presents the investigation and research work carried out for the project. It discusses the domain research related to argumentation theory, argument mining, logical fallacies, and detection. It also examines the application of AI models in debate analysis, including transfer learning, machine learning, and fine-tuning techniques. This chapter also reviews the technical aspects required for system development, including hardware requirements, programming languages, IDEs, libraries, operating systems, and deployment tools.

2.2 Domain Research

2.2.1 Argumentation Theory and Argument Mining

Argumentation, including debates, discussions, and persuasive speeches, is vital in human communication. Hence, the structure of a debate or a speech transcript needs to be broken down and evaluated to understand each sentence's argument meaning further.

2.2.1.1 Toulmin's Argumentation Model

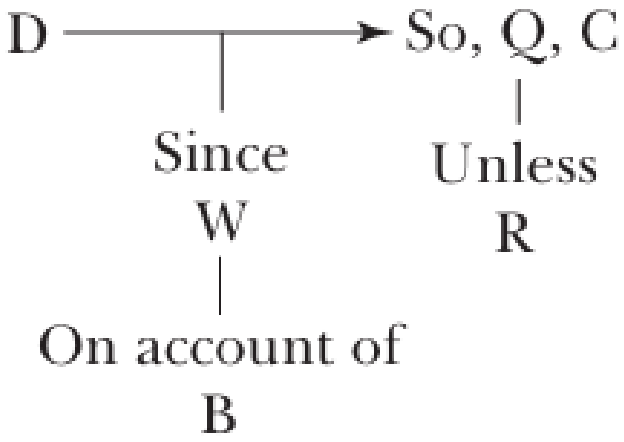


Figure 2: Toulmin's Argumentation Theory (Toulmin, 2003)

In *The Uses of Argument* by Toulmin (2003), the author developed Toulmin's Argumentation Theory, which proposed a structured approach to analysing arguments systematically. This theory breaks down scripts and arguments into identifiable components, providing a clear framework for understanding and assessing the validity of arguments (Verheij, 2005; Khoirunisa & Indah, 2022). Toulmin's Argumentation Theory consist of six fundamental components, which are claims (C), grounds (D), warrant (W), backing (B), qualifier (Q), and rebuttal (R) (Toulmin, 2003).

Claims

The claim is the central assertion or conclusion of the argument (Satishkumar, 2024). It is the statement that the rest of the argument seeks to support. Claims can be explicit or implicit and are often the starting point for analysing an argument (Gupta et al., 2024). Identifying the claim sets the direction for the argument's strength and validity analysis. A strong claim should be well-substantiated and relevant to the topic (Hammond & Charalampidi, 2020; Ruiz-Dolz et al., 2023), which should be clear, specific, and well-supported by the data.

Ground

Ground or data is the evidence of facts presented to support the claim (Mirzababaei & Pammer-Schindler, 2021). Ground can be in various forms, such as statistics, expert opinions, or personal experience. Analysing the grounds helps assess the argument's credibility and relevance. Irrelevant data can undermine the strength of the argument (Li, 2024).

Warrant

Warrants are the underlying assumptions that explain why the evidence supports the assertion. It is often implicit, not stated, and may require inference on the analyst (Amaral et al., 2023). The warrant should be based on valid assumptions and principles, as a weak or unsupported warrant (clarity) can lead to a flawed argument (Hasan et al., 2021).

Backing

Backing refers to the additional support for the warrant. It can include definitions, explanations, or further evidence that strengthens the connection between the ground and the claim (Verheij,

2005), ensuring that the warrant is not based on unfounded assumptions (Satishkumar, 2024). Inadequate backing can make the argument's strength backfire.

Qualifier

Qualifier indicates the strength of the warrant or scope of the claim ("Toulmin Argument," 2023). It is often expressed through words like "most", "usually", "many", or "always". It acknowledges that arguments are rarely absolute and allows for exceptions, which helps to understand the argument's limitations and the degree of certainty (Satishkumar, 2024).

Rebuttal

Rebuttal addresses potential counterarguments or exceptions to the claim (Satishkumar, 2024). It demonstrates awareness of alternative perspectives and strengthens the argument by pre-emptively responding to criticism and evaluating its comprehensiveness and resilience to opposition (Long et al., n.d.).

Applying Toulmin's model in the analysis of debate transcripts allows for a systematic evaluation of the separate components of every argument. Reisert et al. (2015) developed a computational approach toward automatically generating structured arguments by applying Toulmin's Model, extracting relevant claims, evidence, and warrants from different web-based documents. Their method values logico-consistency by overt specification of relations ("PROMOTE" or "SUPPRESS") between argument components, but is plagued by the challenge of accurately capturing fine-grained meanings and implicit arguments. Supriyadi (2023) demonstrated through an experiment that the introduction of the Toulmin Pattern of Argumentation enhanced the critical thinking skills and writing achievements of the experimental group students. This demonstrates the relevance and pedagogical value of this theory, aligning with the project's goal of achieving SDG 4.

2.2.1.2 Argument Mining

Argument mining is the systematic identification, classification, and assessment of arguments in texts written in natural language (Lawrence & Reed, 2019). The primary goal of argument mining is to convert unstructured natural language text into structured forms for analysis, summarisation, or integration into computational argumentation systems (Sahitaj et al., 2024). For quite a long time, researchers in the argument mining community have focused on assessing the strength of arguments presented in natural language (Villata, 2024). Argument mining can automatically summarise arguments, identify biased reasoning, and create new argumentative content for learning platforms, policy-making instruments, and decision-making systems that are assisted by AI (Lippi & Torroni, 2016).

Argument mining starts with identifying and classifying claims, premises, and argumentative statements. It is typically done through supervised machine learning approaches trained on annotated corpora (Peldszus & Stede, 2015). Traditional machine learning methods, such as support vector machines (SVMs) and random forests, employ a broad range of textual features, syntactic, semantic, and lexical features, to successfully identify features of argumentation (Haddadan et al., 2019). Sequence labelling methods, such as conditional random fields (CRFs), have shown their worth in argumentative text segmentation, with accurate and high-quality results (Nguyen & Litman, 2018). Neural network models, specifically CNNs and RNNs, can capture more detailed linguistic patterns and contextual information through dense vector representations, thus enhancing performance in argumentative component classification and relation prediction (Ruiz-Dolz et al., 2021). The approaches described here demonstrate some flexibility and can be classified as hybrid models. The N-SAUR model integrates symbolic logical reasoning and neural approaches in argument structure definition, enhancing accuracy and robustness (Amaral et al., 2023). Recent developments utilising large language models (LLMs) have achieved even higher accuracy levels. Fine-tuning LLMs has improved argument detection tasks, such as argument presence, topic, and stance prediction, making them more suitable for diverse domains and thus chosen in this project (Irani et al., 2025).

Argument mining is today a robust means of analysing and understanding human argumentation in text. Researchers can effectively identify and classify argumentative patterns using suitable

models. The techniques enhance argumentation research and pave the way for real-world applications in education, policymaking, and AI systems.

2.2.2 Fallacies and Detection

In argument analysis, fallacy detection is crucial for determining flawed reasoning that can distort the message conveyed or deceive the audience. Fallacy detection is taken more seriously to help educators and debaters assess the quality of their arguments at scale.

2.2.2.1 Logical Fallacies

Logical fallacies are often embedded in formal logical frameworks and informal argumentation, including affirming the consequent, denying the antecedent, and false dilemmas (Petric, 2020). Such logical fallacies are errors in reasoning that weaken, mislead, or destroy the validity of arguments. There are many different classifications of fallacies. Freeman (n.d.) classifies fallacies into three types: fallacies of irrelevance, as demonstrated by ad hominem; fallacies of presumption, which include hasty false cause; and fallacies of ambiguity, including equivocation.

Fallacy	**Description**
Ad hominem	Attack the person's negative traits or characteristics who states the argument rather than addressing the argument (Cholifah et al., 2024).
Ad populum	Appeals to popular opinion, which many people believe, but not logical reasoning (Archie, n.d.).
Appeal to emotion	Manipulates emotional responses instead of persuading using logic (*The Appeal to Emotion Fallacy: Arguing Through Feelings Rather Than Facts*, n.d.).
Circular reasoning	Make the argument conclusion the same as the premise that leads to a logical loop (Nippold, 2023).
Equivocation	Using a word with multiple meanings ambiguously within an argument leads to confusion (Nippold, 2023).

Fallacy of credibility	Appealing to irrelevant authority means relying on the opinion of an authority figure who lacks expertise in the area (Cholifah et al., 2024).
Fallacy of extension	Also known as the Straw Man fallacy, meaning misapplying a general rule to a specific case or misrepresenting an opponent's argument to make it easier to attack (Schumann, 2019).
Fallacy of relevance	This is also known as a Red Herring, which means introducing irrelevant information to distract from the argument (Nippold, 2023).
False causality	Assuming a causal relationship between two events based solely on their sequence (Nippold, 2023).
False dilemma	Presenting only two or limited options when more are available (Nikolopoulou, 2023).
Faulty generalisation	This is also known as hasty generalisation, which means making broad conclusions based on insufficient or unrepresentative evidence (Nippold, 2023).
Intentional	Interpreting a text based solely on the author's intended meaning without evidence (Mambrol, 2016).

Table 1: List of Logical Fallacies

Friedman's (2024) work demonstrates how logical fallacies sabotage critical thinking and enhance emotional or biased decision-making in discussions, media, and politics. This shows the importance of identifying fallacies in argumentative situations.

2.2.2.2 Logical Fallacies Detection

Manually identifying logical fallacies can be challenging when processing large volumes of text data, such as debate transcripts, speeches, and articles. Researchers have developed various computational methods, including natural language processing (NLP), machine learning, and AI techniques, to address these challenges.

One is structure-based methods that analyse arguments through their logical structures. Such methods involve constructing representations of the argument's logical flow and checking against fallacious patterns of reasoning. Lei and Huang (2024) presented logical structure trees by

constructing hierarchical representations of argument logic flow using constituency trees and logical connectives. The results show an improvement in LLMs' accuracy in fallacy detection and classification. Lalwani et al. (2024) employed another structure-based approach that maps natural language arguments to first-order logic (FOL) and checks them using Satisfiability Modulo Theory (SMT) solvers. It provides exact counterexamples that demonstrate the detected fallacies, achieving an F1-score of 71%.

Knowledge-based approaches leverage external knowledge and structured annotations to enhance the ability of language models to detect logical fallacies with better accuracy. Robbani et al. (2024) employed a template-based annotation approach, which describes the underlying logic of fallacies and offers a systematic way of detecting them. Nevertheless, the performance of their language model encountered difficulties in detecting fallacies, achieving an accuracy level of only 0.47. A hypothetical evaluation framework consists of three stages: detection, coarse-grained classification, and fine-grained classification, which are intended to integrate language models with outside knowledge and explainability mechanisms for better explainability (Sourati et al., 2023). The framework enhances the accuracy in detecting fallacies and offers interpretable explanations by elucidating the advantages and disadvantages of different fallacy types.

Another fallacy detection method is Multitask instruction-based prompting, which addresses fallacy detection as smaller, tractable subtasks to improve generalisability and detection performance in various domains. Alhindi et al. (2022) used this technique on T5-Base and T5-Large to enable recognition of 28 various types of fallacies on four datasets spanning domains, including exams, forums, and essays. This technique outperforms task-specific models like fine-tuned T5, BERT, and GPT-3 in in- and cross-domain setups.

Although specialised classifiers can be trained to identify logical fallacies in particular contexts or data sets, they outperform general language models. Jin et al. (2022) proposed a model sensitive to logical form, which explicitly predicts the logical structure and classifies types of fallacy in a two-stage manner. The model generates a logical form via an auxiliary parser and classifies fallacies depending on this identified structure. This approach achieves a 5.46% F1-score gain on the LOGIC dataset and 4.51% on the LOGICCLIMATE dataset over baseline large language models. These specialised classifiers have real-world uses in tools like Hugging Face. Jin et al.'s (2022) Distilbert-based model introduced a lightweight yet efficient classifier for logical fallacies,

which is suitable for resource-limited settings or rapid deployments. Kanadan (2024) also tuned a RoBERTa-large model to detect multiple logical fallacies on different text genres.

In brief, the evolution of computational approaches to detect logical fallacies has made significant progress, drawing on structure-based analyses, knowledge-driven annotations, multitask prompting, and domain-specific classifiers such as DistilBERT and RoBERTa. These developments facilitate individuals' identification and comprehension of logical fallacies in different contexts, enabling more transparent and effective communication.

2.2.3 AI Models

Finding and training a suitable model is crucial for creating a debate analyser that can successfully analyse and interpret complex linguistic structures, delivering good performance and precise results.

2.2.3.1 Machine Learning and Deep Learning Models

Machine learning (ML) is a field of artificial intelligence (AI) that specialises in the development of systems proficient in learning from data and enhancing their performance without explicit coding (Nilsson, 1998). Deep learning (DL) is a machine learning speciality that entails using neural networks comprising several layers that can independently identify complex patterns in large datasets (Goodfellow et al., 2018).

Support Vector Machines (SVMs)

Support Vector Machines (SVMs) are potent and effective classification techniques designed to suppress the variations between various classes in a high-dimensional context (Cortes & Vapnik, 1995). They continue to be effective even when dealing with small datasets; thus, their significance is in cases where labelled data is hard to find. Singh (2025) conducted a comparative analysis of 64 argumentation mining studies and asserted that traditional models like Support Vector

Machines (SVM) work well for shorter user-generated content. Conversely, top-level deep learning models like BERT and SciBERT demonstrate greater efficacy in more complex contexts like scientific and legal text. Besides, the work by Patel (2023) verifies that SVM remains the highest-performing model employed in argumentation mining on user-generated content with small corpora.

Stab and Gurevych (2017) presented an end-to-end solution for parsing the argumentative structure of persuasive essays by employing Support Vector Machine (SVM) classifiers for identifying argumentative components, determining their types, and predicting their relations, and Integer Linear Programming (ILP) for enhancing the overall model. Their results demonstrated component identification success and robust relation and stance prediction results. Niculae et al. (2017) suggested an open factor graph model for argument mining with structured SVMs and structured RNNs for joint fine-tuning of proposition classification and argumentative relation prediction. Structured models performed better than unstructured baselines and obtained state-of-the-art link prediction performances.

Neural Networks

Artificial Neural Networks (ANNs) are a set of deep learning algorithms inspired by the human brain's structure to replicate complicated and nonlinear relationships (Qamar & Zardari, 2023). ANNs consist of input, hidden, and output layers that identify patterns via weight adjustments, enabling pattern detection, classification, and natural language processing tasks. ANNs possess self-organisation, flexibility, and the ability to operate in real time, enabling the prospective development of different architectures such as feed-forward, convolutional, and recurrent neural networks.

Ruiz-Dolz et al. (2023) presented a hybrid argument mining model to predict the winner of professional debates by combining argumentation theory, Transformer-based sentence embeddings, and graph neural networks. Evaluation on the VivesDebate corpus demonstrated that the hybrid model outperforms formal reasoning-based or NLP-based baselines. Kim (2014) demonstrated that a simple CNN with pre-trained word embeddings, i.e., word2vec, can perform very well in sentence classification with little hyperparameter tuning. Small CNNs can outperform

larger and deeper deep learning models on various natural language processing tasks when trained with these embeddings and used in a multichannel setup. Mao et al. (2023) proposed a Heterogeneous Graph Matching Network (HGMN) designed to recognise pairs of arguments. HGMN consists of a heterogeneous graph attention network (Heter-GAT) for modelling long-distance context and a multi-granularity graph matching model for capturing complex semantic relationships. Experimental results show that HGMN outperforms existing methods and GPT models on peer-review datasets. Zhang et al. (2023) employed Graph Neural Networks (GNNs) with graph augmentation and collective classification for modelling legal documents as computational graphs for argument mining. The approach is evaluated on European Court of Human Rights (ECHR) and Court of Justice of the European Union (CJEU) datasets, where it is shown that ResGCN models with virtual nodes perform better than baseline pipeline approaches, such as RoBERTa and support vector machines (SVMs).

2.2.3.2 Transfer Learning and Pre-trained Models

Transfer learning is a machine learning method that utilises a pre-trained model, i.e., it is already established for one task. It contains relevant knowledge to enhance the learning process for a new task. It is beneficial when there is not much data for the new task (IBM, 2024). The promise that lies in transfer learning minimises the reliance on big data sets and high computing power (Terra, 2024). G. Chen et al. (2023) highlighted how large language models like BERT, GPT-3, and domain-specific models like Legal-BERT have advanced argument mining by improving tasks like argument classification and counterargument generation. It highlights how fine-tuning and prompt engineering are some of the most crucial techniques that significantly advance the performance of LLMs compared to traditional approaches.

BERT (Bidirectional Encoder Representation from Transformer)

BERT was introduced by Devlin et al. (2019). The model can capture deep bidirectional context in understanding language, which has been pre-trained on masked language modelling that predicts missing words and following sentence prediction tasks to be fine-tuned for new tasks.

Shargunam et al. (2024) separately fine-tuned BERT models on the Fake News Challenge Stage 1 (FNC1) dataset for stance detection and the IBM Debater dataset for argument mining. They then combined them into a hybrid model for analysing online discourse. Findings report fine-tuned BERT with 99.92% accuracy, classifying stances and argument structures well. Toledo-Ronen et al. (2020) applied multilingual BERT (mBERT) to stance classification and evidence detection in five languages other than English. Their methods work well for stance and evidence, but not as well for argument quality, since label degradation happens during translation. Dutta et al. (2022) proposed Selective Masked Language Modelling (sMLM) and fine-tuning to RoBERTa and Longformer to improve argument mining with unlabeled online debates in Reddit's ChangeMyView forum. The findings indicate that sMLM has boosted the performance of the models in identifying claims, premises, and their relations. Mushtaq and Cabessa (2023) proposed BERT-MINUS, a modular variant of the BERT architecture that knowingly fine-tunes specific network components to enable knowledge transfer across various tasks. The model obtained state-of-the-art performance in both argument type classification and link detection. Harly and Girsang (2022) merged CNN models with BERT embeddings, referred to as CNN-BERT, and tested their performance by measuring the concordance between arguments using BERT and Siamese-BERT. The results clearly show that the approach enhances accuracy by 3.29% without requiring additional pretraining. Among the primary BERT models, He et al. (2020) identified DeBERTa (Decoding-enhanced BERT with disentangled attention) as a superior option for text classification tasks over BERT and RoBERTa. Its enhanced attention mechanism and embedding representation enable more effective contextual comprehension.

GPT (Generative Pretrained Transformer)

OpenAI developed GPT, a single-directional transformer model that generates the next token in a sequence. Their models are primarily designed for generative tasks and can produce coherent, contextually relevant answers (Ray, 2023). Brown et al. (2020) showed the ability of GPT-2 and GPT-3 to conduct natural language processing tasks like translation, question answering, and cloze tasks in a "few-shot" setup without task-specific fine-tuning.

Mao et al. (2024a) tested the performance of ChatGPT in extracting argument pairs from conversational data based on LLMs under a prompt-based paradigm, namely Multi-Prompt Chain of Thought (MPCoT) and ArgumentAgent. The findings demonstrate that the models perform better than traditional prompt approaches, e.g., Few-shot and CoT-based techniques. Rescala et al. (2024) tested the capabilities of large language models (LLMs), e.g., GPT-3.5, GPT-4, Llama 2, and Mistral 7B, in identifying and evaluating persuasive arguments, predicting one's opinions relying on demographic traits and belief structures, and determining how various arguments persuade various listeners. The findings indicate that while GPT-4 demonstrates high performance, the stacking method—composing outputs of various large language models (LLMs)—yields more favourable results than GPT-4. G. Chen et al. (2024) compare the performance of LLMs, e.g., ChatGPT, Flan, and LLaMA2, on zero-shot and few-shot argumentative component extraction and argument generation tasks. The results highlight the strength of LLMs, especially ChatGPT and Flan, without fine-tuning.

In their study, Yang et al. (2025) fine-tuned ChatGPT to automatically score the scientific explanations of Chinese middle school and high school students, which had been a difficult task before because of the complexity of the language and thought patterns. The results show that fine-tuned ChatGPT scored over 75% on tasks in scoring, thus demonstrating the viability of fine-tuning LLMs for educational assessment. Cabessa et al. (2024) paired kNN-based demonstration instance selection and majority vote ensemble with GPT-4 and fine-tuned GPT-3 for argument type classification (ATC). The outcome demonstrates GPT-4 achieve competitive accuracy under zero-shot conditions, and fine-tuned GPT-3 with textual features designed carefully to achieve state-of-the-art ATC performance. Pietron et al. (2024) paired small models such as DistilBERT and BERT with ChatGPT-4 to support argument classification and confirm uncertain predictions. This hybrid method sends unclear cases to ChatGPT-4, leading to a 10–16% increase in F1 scores compared to using compact models alone.

LLaMA (Large Language Model Meta AI)

Meta AI recently presented LLaMA models, open alternatives to bigger proprietary models like GPT-3 and PaLM (Touvron et al., 2023). LLaMA exhibits strong performance in zero-shot, few-shot, reasoning, code generation, and reading comprehension tasks, focusing on bias, toxicity, and carbon footprint concerns. Gorur et al. (2024) assessed the capability of large language models (LLMs) to carry out relation-based argument mining. The findings indicate that Llama-2 and Mistral perform better than RoBERTa-based baselines in recognising support and attack relations in arguments.

Bao et al. (2024) presented a Comparison-Based Argument Quality Assessment (CompAQA) framework that tailors BERT, RoBERTa, DeBERTa, and LLaMA-3-8B for comparing arguments for better structuring and relation analysis. The results show that the fine-tuned CompAQA-Llama decoder and CompAQA-DeBERTa encoder-only models performed very well. Mao et al. (2024b) introduced a generative model based on the Llama2-7b model, named LLaMA Argument Pair Extractor (LAPE). LAPE effectively mitigates error propagation, captures complex argument structures, and achieves high performance compared to traditional extractive approaches.

DeepSeek

DeepSeek is an open-source language model for structured reasoning and data retrieval (Sen, 2025). It was launched in 2023, specialising in tasks like mathematical reasoning and coding that emphasise real-time learning and adaptability. It also positioned itself as a strong competitor in democratising AI, although it faced challenges with consistency and market recognition (Shah et al., 2025). Gao et al. (2025) compared DeepSeek-R1 with Claude, Gemini, GPT, and Llama on authorship and citation classification tasks, showing that DeepSeek outperforms other models but slightly underperforms Claude in accuracy. DeepSeek offers strong prediction performance at a much lower cost despite being slower.

Rhomrasi et al. (2025) fine-tuned DeepSeek-R1 and evaluated its ability to solve high-school-level mathematical reasoning problems in Catalan from the Kangaroo Mathematics Competition. The results show 96% of model accuracy, outperforming GPT, Gemine, and ALIA (Spanish LLMs

trained with Catalan exposure) models and demonstrating the effectiveness of reasoning-focused fine-tuning. Although research on fine-tuning DeepSeek remains limited, it shows promising potential for enhancing debate analysis, with several practical cases and tutorials available on reputable platforms such as DataCamp (*Fine-Tuning DeepSeek R1 (Reasoning Model)*, 2025), Medium (Nasir, 2025), and various community-driven fine-tuning DeepSeek projects available on GitHub and Hugging Face.

To build the debate analyser, this project trains and compares six model families for argument-type classification, including SVM, a lightweight CNN, and four transformers, which are DeBERTa, GPT, DeepSeek, and TinyLLaMA. The transformer variants are taken from publicly available checkpoints that can be run on a laptop. Because computing resources are limited and some research provides few publicly transparent results, the study should run a consistent pipeline across all models to produce clear, reproducible benchmarks. Fallacy detection is not retrained in this work, since resources are constrained, and some reliable pretrained detectors from Hugging Face can be applied directly.

2.2.4 Bias Mitigation in Language Models

LLMs trained on massive textual data scraped from the web and other large-scale sources (Bender et al., 2021) can reflect societal imbalances, stereotypes, historical biases, and others (Mehrabi et al., 2021) that cause models to internalise and produce biased behaviour through downstream tasks (Gallegos et al., 2023). These biases can directly impact the outcomes of argument analysis and misrepresent the voices of underrepresented groups (Holtermann et al., 2022). Thus, it is necessary to research bias mitigation in LLMs for fairness. Gallegos et al. (2023) had categorised bias mitigation methods into three categories: pre-processing, in-training, and post-processing.

The pre-processing step cleanses and balances the training data before training the models. One of the most popular methods is counterfactual data augmentation (CDA), which involves generating replacement text variants to de-bias stereotypical associations (Lauscher et al., 2021; Zhao et al., 2021). For example, gendered pronouns (“he” to “she”) and profession references are swapped in

training samples to ensure balanced exposure. In comparison, in-training methods involve modifying the model's learning process. The study by Gallegos et al. (2023) used adversarial training to mitigate bias by introducing an adversary model that attempts to predict sensitive attributes from the main model's internal representations, while the main model is trained to prevent this prediction. This process forces the model to learn feature representations that are less dependent on protected attributes, promoting fairness during training. Another method is adapter-based debiasing, by introducing a lightweight neural module (adapter) into a pre-trained LLM and fine-tuning this module to reduce biased outputs (Holtermann et al., 2022). It improved fairness in tasks such as argument type classification and fallacy detection without degrading performance to reduce positional, gender, and tone-based biases. Post-processing methods adjust model outputs after generation. Y. Zhang and Sang (2024) developed Inference-Time Rule Eraser (IRE), which removes biased patterns from model predictions without retraining. It is helpful in low-resource settings and when models must be updated quickly without complete retraining cycles. Li et al. (2024) used a causality-guided prompt design that frames input prompts in ways that guide the model away from biased associations. It is effective with instruction-following models, such as ChatGPT, where the prompt structure has a significant influence on the output.

In argument mining, bias mitigation strategies help ensure that arguments are evaluated on the grounds of logical form, rather than being influenced by irrelevant biases. Holtermann et al. (2022) suggested the ABBA corpus to test bias in computational argumentation. They reported that LLMs generate a preference for arguments with certain linguistic or stylistic features, leading to biased scoring without mitigation. Their adapter-based debiasing research demonstrated that incorporating fairness-specific modules into argument quality rating models can mitigate biases. Qureshi et al. (2024) employed reinforcement learning to fine-tune fairness objectives in language generation. It is a promising direction to debias models used in evaluative tasks.

These findings demonstrate that argument mining systems must incorporate bias mitigation strategies to maintain fairness and reliability. These techniques help the model focus on the quality of the argument, allowing debate systems to provide fairer feedback and encourage more balanced discussions. This supports SDG 4 by providing equal and high-quality education for everyone.

2.3 Similar Systems / Works

Journal / Paper Title	Author(s)	Description	Dataset Used	Method	Evaluation Techniques	Results / Outcome	Limitation
Argument Mining with Modular BERT	Mushtaq and Cabessa (2023)	The paper introduces a modular BERT-based model called BERT-MINUS, designed to classify argument types and identify argumentative links. It uses contextual, structural, and syntactic features as input, processes these with dedicated BERT modules, and applies a novel Selective Fine-tuning method for intra- and inter-task transfer learning between argument	Persuasive Essays (PE) dataset	BERT-MINUS architecture with joint and dedicated BERT modules, Features-as-Text for enriched embedding, and Selective Fine-tuning for auto-transfer and cross-transfer learning	Macro F1 scores on Argument Type Classification and Link Identification tasks, compared against baselines including standalone BERT and earlier models	With Features-as-Text and cross-transfer fine-tuning, BERT-MINUS achieved state-of-the-art results on Link Identification and competitive results on Argument Type Classification, showing benefits of modularisation and transfer learning.	Model performance depends heavily on the quality of hand-crafted feature texts; the system still underperforms compared to some joint-learning models in Argument Type Classification without full task-specific tuning.

		classification and link prediction.					
In-Context Learning and Fine-Tuning GPT for Argument Mining	Cabessa et al. (2024)	The paper presents a strategy that uses both in-context learning (ICL) and fine-tuning GPT models to classify essay argument types. It explores how different prompting strategies and feature additions influence model performance.	Persuasive Essays (PE) dataset with 402 essays annotated with argument components: Major Claim, Claim, and Premise.	The authors use a two-step ICL strategy combining kNN-based example selection and ensembling with GPT-4, and fine-tune GPT-3.5 using contextual and structural features in textual form.	Macro F1-score across three argument types (Major Claim, Claim, Premise); comparison of ICL and fine-tuning setups; and effect of additional features.	The ICL strategy with GPT-4 achieves an F1-score of 83.6%, outperforming fine-tuned BERT variants. Fine-tuned GPT-3.5 with structural features reaches 86.3%, setting a new state-of-the-art for single ATC tasks.	Adding too many explicit structural features may reduce performance. GPT-3.5 underperforms compared to GPT-4 in ICL, showing limitations of smaller models in training-free setups.
Leveraging Small LLMs for Argument Mining in Education	Favero et al. (2025)	This paper explores how small, open-source language models like Qwen 2.5 7B, Llama 3.1 8B, and Gemma 2 9B can be	Feedback Prize - Predicting Effective Arguments (a subset of the PERSUADE 2.0	The study used two approaches: few-shot prompting and fine-tuning of small LLMs. It	Evaluation was based on macro-averaged F1 scores for segmentation,	Fine-tuned small LLMs outperformed the state-of-the-art baselines. Llama 3.1 8B achieved	The models struggled with argument quality assessment due to inconsistent labelling in the

		used for argument mining in student essays. It focuses on three tasks: identifying argument segments, classifying their type, and assessing argument quality. The goal is to provide accessible, real-time feedback to help students improve writing skills using local, low-resource tools.	Corpus), containing 6,900 essays from students in grades 12 with 26,000 annotated argument components.	tested open-source models against baselines (Longformer and BERT) and commercial GPT-4o mini. Tasks were done individually and jointly using segmenting, argument type labelling, and quality scoring with custom prompts and JSON outputs.	argument type classification, and quality assessment. BIO-tagging was used for segmentation. Results were compared against gold and inferred segmentation.	87.52 F1 in segmentation (26% better than Longformer). Gemma 2 9B scored 79.74 F1 in type classification (14.78% better than BERT). Quality assessment was less accurate, with the best model (Gemma 2 9B, few-shot) scoring 44.56 F1.	dataset. Also, some fine-tuned models failed in individual setup tasks. The dataset was limited to English essays by US students, so findings may not generalise to other languages or contexts.
Argument-Based Detection and Classification of Fallacies	Goffredo et al. (2023)	This paper presents a method for detecting and classifying logical fallacies in political debates using	An extended version of the ElecDeb60to16 dataset, which includes US	The authors propose a MultiFusion BERT architecture that combines	The model was evaluated using accuracy, F1-score, and	MultiFusion BERT outperformed other models, showing a 2.12%	The dataset was limited to US political debates, making generalisation to

		argument-based features. It improves existing models by integrating argumentative component annotations into a Transformer-based architecture, enhancing the model's ability to recognise six fallacies, such as ad hominem and false causality.	presidential debates from 1960 to 2020, annotated with argument components and six fallacy types. It was expanded with debates from the Trump-Biden era.	textual inputs with argumentation-related features like component type and argumentative role. The model uses Transformer layers for encoding and fusion mechanisms to enhance fallacy prediction.	confusion matrices for fallacy classification. The results were compared with those of baseline models, including vanilla BERT and RoBERTa.	improvement in classification performance. The fusion of argumentative features helped the model better understand the structure and context of fallacies.	other domains difficult. Also, the model's high GPU requirements pose challenges for wider practical use.
Parsing Argumentation Structures in Persuasive Essays	Stab & Gurevych (2017)	End-to-end parser that identifies components and jointly optimises component types and argumentative relations with an ILP joint model on top of base SVM classifiers.	Persuasive Essays corpus, 402 essays.	Sequence labelling for component spans, SVM base classifiers for types and links, global ILP inference to enforce well-	Macro-averaged precision, recall, and F1; significance via Wilcoxon;	ILP joint model improves component F1 and linked-pair F1 over base classifiers; component F1 approaches	Genre sensitivity and pipeline limits and essays are explicit, while other genres include implicit arguments,

				formed trees and stance.	component, relation, and stance sub-tasks.	human upper bound.	making transfer harder.
Argument Mining with Structured SVMs and RNNs	Niculae, Park & Cardie (2017)	Factor-graph model that jointly predicts proposition types and links, with structured SVM and structured RNN parametrisations and higher-order constraints.	UKP argumentative essays and CDCP e-rulemaking comments.	Linear structured SVMs and structured RNNs with constraints and higher-order factors; joint inference for types and links.	F1 at the proposition and link level, and their average; k-fold CV at the document level.	Structured learning outperforms unstructured baselines; it improves UKP link prediction over prior ILP post-processing.	On CDCP, baseline inference is not competitive; feature-engineered SVMs still outperform RNNs on links, highlighting the difficulty of non-tree debates.

Table 2: Similar Systems / Works

2.4 Technical Research

2.4.1 Hardware Requirement

Component	Specification
Processor (CPU)	13th Gen Intel(R) Core(TM) i7-13620H 2.40 GHz
Graphics Card (GPU)	NVIDIA GeForce RTX 4050 Laptop GPU
Random Access Memory (RAM)	32 GB
Storage	512 GB
System Type	64-bit operating system, x64-based processor

Table 3: Hardware Equipment

This project will be developed and tested on a personal laptop that meets all the requirements for running Python, training machine learning models, and handling data processing tasks. The hardware should be enough to support local development, testing, and light model training. If the hardware is insufficient, heavier tasks, such as fine-tuning large models, will be run on cloud platforms.

2.4.2 Programming Language

Python is a high-level, versatile programming language that has become one of the most popular choices for artificial intelligence (AI) and data science projects. It was first released in 1991 by Guido van Rossum to emphasise code readability and simplicity (GeeksforGeeks, 2025a). Python supports multiple programming paradigms, including object-oriented, procedural, and functional programming, making it highly flexible for different applications (GeeksforGeeks, 2025c). It also has a large and active community that continuously contributes to its ecosystem. It offers thousands of libraries and frameworks that simplify complex tasks, such as machine learning, natural language processing, and web development (Hossain, 2024).

	Python	JavaScript	C++	Ruby	PHP	Lisp	Java
Syntax	Simple	Complex	Complex	Simple	Simple	Flexible	Complex
Popularity	Wide	Popular for web	System coding	Popular for web	Popular for web	Niche language	Enterprise-level apps
Typing	Dynamically typed	Dynamically typed	Statically typed	Dynamically typed	Dynamically typed	Dynamically typed	Statically typed
Performance	Slower than C/C++	Slower than C/C++	Fast	Slower than C/C++	Slower than C/C++	Fast when optimized	Fast
Community	Vast and active	Vast and active	Active	Active	Active	Niche	Vast and active
Learning Curve	Beginner-friendly	Moderate learning	Steeper learning	Beginner-friendly	Moderate learning	Steeper learning	Moderate learning
Use Cases	Web dev. data analysis, AI	Web,mobile app, game dev.	System software, game dev., performance-critical apps	Web dev., automation, scripting	Web dev., server-side scripting, creating dynamic web pages	Symbolic processing, AI, numerical analysis, prototyping	Enterprise-level apps, Android and web dev
AI Dev. Suitability	Widely used for AI programming	Used	Used in performance-critical parts	Less prevalent	Limited use	AI symbolic processing & prototyping dev.	Niche
Code Length	Short	Moderate	Long	Moderate	Moderate	Moderate	Moderate
Code Bases	Large and diverse	Extensive code bases for web dev.	Extensive niche code bases	Extensive for web dev.	Extensive code bases for web dev.	Small	Extensive niche code bases
Legacy	Continuously evolving	Continuously evolving	Many legacy codebases	Many legacy codebases	Many legacy codebases	Legacy in niche apps	Many legacy codebases
Sample Code	print("Hello, Unimates")	console.log("Hello, Unimates");	std::cout << "Hello, Unimates";	puts "Hello, Unimates"	echo "Hello, Unimates";	(print "Hello, Unimates")	System.out.println("Hello, Unimates");

Figure 3: Python Compared to Other Programming Languages (Marga, 2024)

Compared to other programming languages like Java or C++, Python offers faster development speed and lower complexity for AI applications. Its syntax is closer to human language, making it easier to prototype and iterate machine learning models quickly. This is especially beneficial in academic research, where both efficiency and accuracy are critical (Marga, 2024).

Python is used for this project since it caters to every stage of developing an AI-driven debate analyser from data preprocessing, modelling, evaluation, and deployment. Python offers simple access and great libraries like Pandas for data processing and preprocessing, NLTK and spaCy for natural language processing, scikit-learn for machine learning model development, and TensorFlow or PyTorch for deep learning models (GeeksforGeeks, 2024). For interacting with Large Language Models such as BERT, GPT, and LLaMA, libraries such as Hugging Face Transformers provide simple APIs for inference and fine-tuning (Wolf et al., 2020). The second reason Python was chosen is the way it integrates so well. Libraries such as Flask or FastAPI provide simple deployment of machine learning models into simple web applications quickly, providing a simple-to-use interface where the user can upload transcripts to be processed. Python also possesses visualisation libraries like Matplotlib and Plotly for creating dashboards (Trivedi, 2024) that users can use to visualise the argument structures, fallacy detection, and system recommendations more intuitively.

In conclusion, Python's simplicity, extensive libraries, strong community support, and ease of integration are reasons enough for it to be the best for creating a debate analysis system that meticulously and comprehensively addresses all phases, ranging from data preparation to interactive deployment.

2.4.3 Interactive Development Environment (IDE)

Visual Studio Code (VS Code) is a free, lightweight, and extensible code editor developed by Microsoft that can run on Windows, macOS, and Linux (Heller, 2022). Its modern interface and text editing come with features like intelligent code completion (IntelliSense), syntax highlighting, debugging tools, integrated terminal, seamless version control via Git, and increasingly (Adak, 2024). Besides, it also supports Jupyter Notebooks, which allow users to run notebook code cells directly within the editor, with additional features such as variable inspection, cell execution, interactive debugging, and more (*Jupyter Notebooks in VS Code*, 2021). VS Code is flexible, allowing easy switching between Python scripts and notebooks in a unified environment that streamlines the development experience.

IDE / Editor	Key Strengths	Best Suited For
Visual Studio Code	Lightweight, highly customizable with rich extensions (Python, Git, debugging, terminals)	General-purpose development with flexibility
PyCharm	Full-featured IDE with advanced debugging, code analysis, and web framework support	Professional development and large projects
IDLE	Comes pre-installed with Python, lightweight, basic syntax highlighting, debugger	Quick scripting, beginners with minimal setup
Jupyter Notebook	Interactive, cell-based execution; ideal for exploration and visualisation	Experimentation, prototyping, presentation
Replit	Browser-based, no installation needed; real-time collaboration, built-in hosting, supports many languages	Team projects, education, and rapid prototyping online
Spyder	Built for scientific workflows; integrates tools like variable explorer, data output	Data science and exploratory analysis
Thonny	Beginner-friendly, simple UI, step-by-step debugging, variable visualisation	Teaching and early learners

Table 4: Python IDEs Comparison Table (GeeksforGeeks, 2025e)

Other IDEs and development environments each have their own strengths. For example, PyCharm is introduced by JetBrains, which is an IDE with rich features tailored for Python development, with advanced refactoring, code analysis, graphical debugging, built-in testing tools, and tight integration with virtual environments, but some features require a professional version subscription (Pushkar, 2025). While IDLE is the standard Python editor bundled with Python, it is simple, lightweight, and ideal for quick scripting, but lacks advanced tooling (Fadilpašić, 2025). Replit is an online browser-based IDE for rapid prototyping and collaboration without local setup (Graham, 2025). There are many other platforms like Spyder, Thonny, and Eclipse with their special features targeted for different users and project purposes.

For this project, VS Code is chosen due to its versatility and light footprint, which enable better management on both exploratory and production stages of development, and most importantly, it costs nothing. Its integrated features, such as debugging tools, IntelliSense for code assistance,

version control, integrated terminal, and workspace flexibility, reduce context switching and enhance productivity (Sandu, 2025). Remote development is also simplified via built-in extensions like Remote–SSH and containers, which offer the flexibility to connect to cloud-based and remote development environments when needed (Malaviarachchi, 2024). Plus, the native support for Jupyter Notebooks ensures smooth transitions between data exploration and modelling phases, while Python scripts can be maintained cleanly for building the user interface and deployment logic.

VS Code with Jupyter Notebooks for interactive data analysis, visualisation, and common modelling is ideal. However, for heavy modelling like pre-trained models and deep learning frameworks, whose local hardware specs are hard to afford and insufficient, cloud platforms like Google Colab will be considered because it is also a cloud-based Jupyter environment with free access to GPUs and TPUs (Burke, 2023). VS Code complements this by allowing notebooks to be developed locally for refinement, then executed in Colab when extra computational power is needed. The notebook's ability to allow tolerance for error, which means each code cell is separate and no need to rerun all script during an exception, unlike other IDEs. This hybrid setup combines convenience, interactivity, and scalability.

In conclusion, VS Code with Jupyter Notebooks is the most suitable and balanced for this project in all stages, from data processing to deployment. Other IDEs are also excellent, but the flexibility, extension ecosystem, lightweight nature, and support for notebooks of VS Code make it the choice. With the fallback option of Google Colab, it gives a guarantee on the development flow to complete heavy modelling.

2.4.4 Libraries

Several Python libraries will be used in this project to develop the AI debate analyser. Suitable libraries can make development faster, more reliable, and more efficient. The libraries will be used throughout the process, such as cleaning and preparing text data, training and evaluating models, and handling argument structures. The Python libraries and their usages can be found from The Python Package Index (PyPI) (https://pypi.org/), a Python software repository providing the introduction and guidance for every library.

The table below lists the main libraries that could be used in this project.

Library	**Description**
Pandas	Pandas is a Python library for data manipulation and analysis. It will be used during preprocessing to clean, structure, and organise the debate transcript data.
NumPy	NumPy is a library that supports large, multi-dimensional arrays and matrices. It will handle numerical operations and optimise data storage during feature engineering.
NLTK	NLTK is a natural language processing library that provides tokenisation and text cleaning tools. It will be used to preprocess the sentences by removing stopwords and punctuations.
spaCy	spaCy is an NLP library designed for advanced text processing. It will help to do tokenisation, lemmatisation, and part-of-speech tagging on debate transcripts.
Scikit-learn	Scikit-learn is a machine learning library for Python. It will be used to build traditional models like SVM and evaluate them with metrics such as accuracy and F1-score.
TensorFlow	TensorFlow is a deep learning framework developed by Google. It will be used to build and fine-tune deep learning models for argument classification and fallacy detection.
PyTorch	PyTorch is another popular deep learning library. It is recommended for training neural network models, especially if more flexibility is needed in model building.
Hugging Face Transformers	Hugging Face Transformers is a library for working with pre-trained language models. It will be used for logical fallacies labelling, transfer learning, and fine-tuning LLMs like BERT, GPT, and LLaMA on the debate dataset.
Matplotlib	Matplotlib is a visualisation library for creating basic plots, such as model performance graphs and data distribution charts.
Seaborn	Seaborn is a Python data visualisation library based on Matplotlib. It will be used to create more detailed graphs to understand model results.

Plotly	Plotly is an interactive visualisation library. It helps create interactive charts that allow users to explore argument structures and detect fallacies.

Table 5: Libraries to be used in Project

2.4.5 Tools

Streamlit is a simple and powerful open-source framework that allows developers to build interactive web applications directly from Python scripts (*Streamlit • a Faster Way to Build and Share Data Apps*, n.d.). It was designed specifically for data science and machine learning projects, making it easy to create clean, functional user interfaces without complicated front-end development like HTML, CSS, or JavaScript (*Code Snippet: Create Components Without Any Frontend Tooling (No React, Babel, Webpack, Etc)*, 2021). With Streamlit, users can create text inputs, buttons, sliders, and display tables, charts, and model predictions in just a few lines of code. Its features to quickly turn data analysis or machine learning models into web apps have become popular for projects requiring fast prototyping and simple deployments (*What Is Streamlit: All Why's and How's Answered | UI Bakery Blog*, n.d.).

Streamlit is easier and faster to set up than web development frameworks like Flask or Django. Flask and Django offer more control and flexibility. Still, they require more backend and frontend coding, which slows development (Andres, 2024), as this project requires a simple interface. Frameworks like Flask also require managing routes and templates and integrating additional libraries to create and style web pages. Streamlit focuses entirely on simplicity. Developers only need to focus on their Python code and logic, and Streamlit handles the web application part automatically.

Streamlit becomes the best choice for this project because the debate analysis system is designed to be simple. The system's primary purpose is to enable users to input a debate or speech transcript, receive evaluations of argument strength, detect fallacies, receive suggestions for improvement, and view descriptive analysis through charts. Streamlit can easily create a text input box, displaying the model's outputs and charts. Streamlit also supports integration with important Python libraries such as Hugging Face Transformers, Pandas, Matplotlib, and Plotly (*Using HuggingFace With Streamlit*, 2024), which are planned for use in this project. Streamlit

applications can also be deployed easily on Streamlit Cloud, Google Cloud, or locally without complicated server setups (*Deploy - Streamlit Docs*, n.d.). Streamlit can help this project focus more on delivering a functional and user-friendly system without spending much time on web development.

2.4.6 Operating System

Windows 11 will be used as the operating system. Windows 11 provides a stable and user-friendly environment that supports Python programming and the development tools needed for this project. It allows the installation of various Python libraries through package managers like pip. Popular IDEs like PyCharm and tools like Streamlit can also run smoothly on Windows 11. Thanks to its strong integration with modern hardware, efficient resource management, and broad compatibility, Windows 11 is a reliable platform for model development and system deployment in this project.

2.5 Summary

Chapter 2 provided a literature review that forms the basis upon which the AI-based debate analyser is developed. It began by examining the key elements of argumentation theory and logical fallacies, presenting Toulmin's Argumentation Model to understand the composition and sequence of an argument. Every element of the model was discussed and connected to its application in debate analysis, such as claim, ground, warrant, backing, qualifier, and rebuttal. The discussion of logical fallacies added depth by identifying common reasoning fallacies, such as ad hominem, false dilemma, or circular reasoning, that compromise arguments and mislead the audience. These theories are necessary to construct a system that can effectively understand and gauge argument quality.

It proceeded to the second area, argument mining and strength assessment, an essential activity in natural language processing and analysis. It discussed argument mining based on component identification, type classification, relation extraction, and quality assessment. It also mentioned existing models and frameworks like CompAQA and SPARK, and how these improve argument comprehension and assessment. This was succeeded by a more specific exploration of argument

strength analysis, in which research illustrated the influence of linguistic structure, rhetorical strategy, and logical coherence on an argument's persuasiveness and strength. This renders the system fallacy-sensitive and enables it to provide constructive feedback on the strength of each argument.

The chapter then discussed AI language models applied in argument analysis. The classical machine learning methods, such as SVM, and the newer state-of-the-art methods, such as neural networks and transformers, were discussed. Pretrained models such as BERT, GPT, LLaMA, and DeepSeek were analysed for classification, fallacy detection, and strength prediction capabilities. The concept of transfer learning was introduced to show how large models can be fine-tuned for specific tasks using little data. After comparing the different models, the application of neural networks, BERT, GPT, and LLaMA was justified based on high performance, flexibility, and availability of community support through forums like Hugging Face.

Bias mitigation in AI systems was another essential part of the literature review. Since unintended biases can influence argument evaluation in training data, the chapter outlined several mitigation techniques. These included pre-processing strategies like counterfactual data augmentation, in-training methods like adversarial training and adapter-based learning, and post-processing methods like inference-time rule erasers and prompt steering. The relevance of these techniques to argument mining and strength evaluation was also discussed, as they ensure fairness and objectivity in the analysis.

Finally, Chapter 2 reviewed similar systems and recent research that align with this project’s goal. Each system was analysed based on the dataset used, methods applied, and outcomes achieved. This comparison helped identify the research gap and inspired potential directions for this project. The chapter closed with technical research evaluating suitable tools, programming language, IDE, libraries, operating system, and dashboarding frameworks for system development. This chapter's domain and technical research provide a solid groundwork for implementing an efficient and reliable AI debate analyser aligned with SDG 4, promoting education and critical thinking.

Chapter 3: Methodology

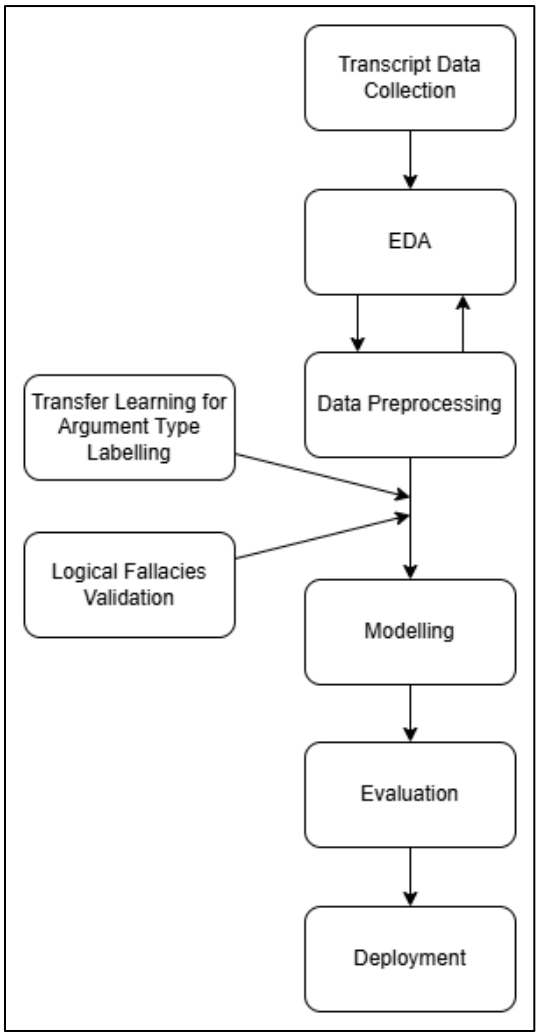


Figure 4: Proposed Method Pipeline

The development of the debate analysis system follows a structured pipeline that begins with collecting debate transcripts from the IBM Debater dataset, which consists of thousands of .txt files. These raw transcripts are combined, converted into a CSV file, and separated into individual sentence-level entries. Every sentence is supplemented with metadata from the filename. Following exploratory data analysis (EDA), there is a cleaning and text preprocessing step. This involves punctuation restoration, sentence splitting to ready the data for machine learning. A Hugging Face transformer-based model is then used to annotate logical fallacies on every sentence. The data is manually sampled to around 5% and utilised for argument type labelling using tools for implementing Toulmin's argumentation scheme. The labelled data is applied to fine-tune the

model with transfer learning, so that the model automatically labels argument types on the entire dataset. The system goes to the modelling phase, where classification models are trained and tested against metrics. Lastly, the system is deployed with Streamlit to develop an interactive user interface. The interface accepts transcripts as input from the user and provides sentence-level insights, including argument flow analysis, fallacy detection, improvement suggestions, and visual analytics.

3.1 Introduction

This chapter outlines the methodology for developing the AI-powered debate analyser, demonstrating the value of following a clearly defined process throughout the project. It begins by introducing and explaining the methodologies to be considered, including CRISP-DM, SEMMA, and KDD, which each provide structured guidance for projects. The most suitable methodology is identified and justified for this project. Each phase of CRISP-DM is outlined to direct the system development.

3.2 Methodology

3.2.1 Introduction to Methodology

A methodology for a data science project is necessary, as it serves as a formal roadmap that assists in structuring, collecting, analysing, and interpreting data. The systematic data mining framework gives clear and structured guidance, rendering study findings more trustworthy (M. Kim, 2021). It can structure each step to minimise errors and ensure the project's adherence to its objectives. A clearly defined methodology also enables teamwork within a group, ensures traceability of work, and promotes overall productivity throughout the project life cycle. CRISP-DM (Cross-Industry Standard Process for Data Mining), SEMMA (Sample, Explore, Modify, Model, and Assess) and KDD (Knowledge Discovery in Databases) are a few of the well-known data mining methodologies.

CRISP-DM is one of the most widely used data science methodology frameworks that various industries can easily adapt. It comprises six primary phases: business understanding, data understanding, data preparation, modelling, evaluation, and deployment. CRISP-DM is widely used due to its flexibility and non-linearity, allowing for backtracking to previous stages when necessary (Smart Vision Europe, 2020). This makes it practical for real-world projects because unknown new insights or data issues can arise during the process.

SEMMA is a methodology developed by SAS Institute, which stands for Sample, Explore, Modify, Model, and Assess. It is a structured approach that can discover meaningful patterns in large datasets (*SAS Help Center*, n.d.-a). It focuses on the technical steps of data mining, like model development. SEMMA has focused less on business understanding than CRISP-DM. Still, it is well-suited for predictive modelling tasks such as fraud detection, customer retention, market segmentation, and risk analysis through SAS Enterprise Miner (*SAS Help Center*, n.d.-b).

KDD is another data mining process model that includes selecting, preprocessing, transforming, and mining data to extract valuable knowledge (GeeksforGeeks, 2025b). It focuses on discovering patterns and insights from large datasets, providing a strong theoretical foundation. KDD is more research-oriented and less commonly used in modern machine learning workflows than CRISP-DM or SEMMA.

3.2.2 Methodology Choice and Justification

CRISP-DM methodology is chosen for developing a debate analyser project because it provides a structured and flexible approach that guides the project through its six clear phases. CRISP-DM emphasises the early understanding of project goals and business context to ensure the data mining work is aligned with real-world objectives (Gómez Palacios et al., 2017). It is more comprehensive than SEMMA, as it includes a business understanding phase to ensure tasks are correctly aligned with the project's overall objectives and goals, as well as a deployment phase for the debate analyser system. This keeps the entire project on track. SEMMA focuses mainly on the technical modelling stages and assumes that the business issue is defined (Azevedo & Santos, 2008), something that may not be possible in this project with a definite definition of objectives and interpretation. CRISP-DM also provides better documentation, actual examples, and explicit task

breakdowns, so users can easily follow step by step (Palacios et al., 2017). In comparison to these, the KDD process is abstract and general. KDD provides a high-level picture of pattern discovery in data but lacks detail on how to apply the steps to real-world projects (Azevedo & Santos, 2008). CRISP-DM is an extension of KDD, providing more detailed steps and strong feedback loops among phases, making it more adaptable to data preparation and modelling issues.

3.2.3 Phases of CRISP-DM

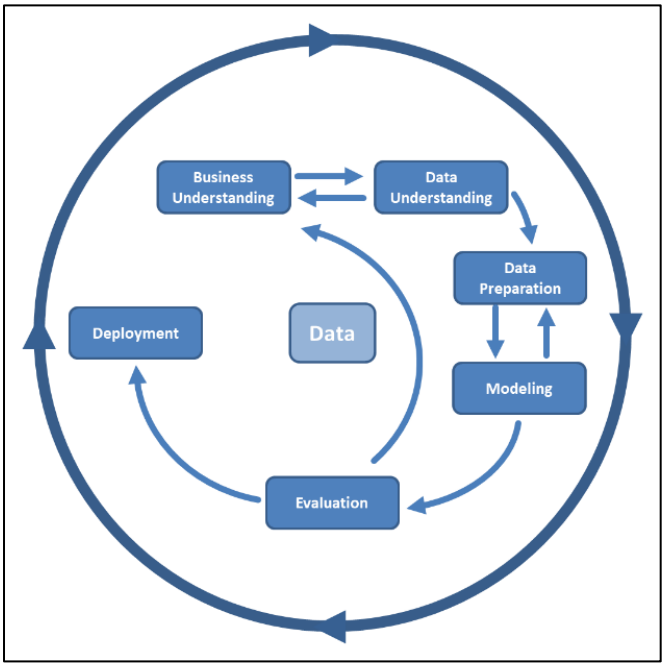


Figure 5: CRISP-DM Life Cycle (Heinrichs, 2025)

3.2.3.1 Business Understanding

During the initial phase of CRISP-DM, it is crucial to have a clear idea of the project's goals, problems, and necessities. This project focuses on creating an AI-based debate analyser that can recognise argument types and logical fallacies and give constructive feedback for improvement using users' debate or speech transcripts. This phase involves reviewing existing issues, including manually examining complex arguments and mitigating the risk of human judgment bias. The objectives and scope of the project were clearly defined to ensure that the chosen methodologies and frameworks align with practical needs. The system is created with its end users—specifically

debaters, teachers, and judges—in mind, as they would benefit from accurate, accelerated, and unbiased feedback. This phase ensures that the development is kept focused on solving real-world problems.

3.2.3.2 Data Understanding

The data understanding phase aims to provide a clear overview of the dataset before pre-processing. Here, the thousands of .txt files of the IBM Debater dataset, with each file being an actual debate transcript, are first loaded and combined into one CSV file via Python. First, exploratory data analysis (EDA) is done at the transcript level to search for possible issues such as missing punctuation or inconsistent formatting. Data cleaning is employed to improve quality. It is then expanded into sentence-level entries with additional metadata such as sentence length, position, and stance. EDA is again performed to examine linguistic and structural patterns at the sentence level using descriptive statistics and visualisation tools such as word frequency plots and sentence length distributions. This step is highly intertwined with data preparation and informs future modelling steps.

3.2.3.3 Data Preparation

The data preparation phase aims to produce a clean, structured dataset for all modelling tasks, including argument type classification and fallacy detection. This process begins with cleaning the raw debate transcripts at the transcript level to address problems identified through EDA. After this, the data is segmented into sentence-level entries, each enriched with metadata from the filename. Further processing is applied at the sentence level to improve accuracy. Text preprocessing techniques are used to normalise the content for machine learning applications. A transformer-based fallacy detection model from Hugging Face is applied to identify and label logical fallacies for model validation purposes, with results stored in a dedicated column. Finally, a representative dataset sample is manually labelled with Toulmin argument types using ChatGPT as an assistant. This labelled sample will later be used for training argument classification models.

3.2.3.4 Modelling

The modelling phase is dedicated to model development and assessment to determine the best argument type classification and fallacy detection method. The phase begins by employing transfer learning to classify argument types across the entire dataset, using the manually labelled sample as training data. Following the labelling of the entire dataset, various models, including SVM, CNN, DeBERTa, GPT-2, LLaMA, and Deepseek, with LoRA techniques if needed, are individually trained on pre-processed debate data. Each model learns to classify argument types and recognise logical fallacies from extracted features. Key hyperparameters like batch size, learning rate, and the number of epochs for training are adjusted for better performance. When the model outcomes are poor, the flow can revert to data preparation to refine features or enhance the quality of the input, and then retrain.

3.2.3.5 Evaluation

The evaluation stage assesses the performance of models trained in terms of argument classification and the identification of logical fallacies. The performance will be assessed using standardised metrics such as accuracy, precision, recall, and F1-score. Additionally, confusion matrices will be generated to visualise the models' prediction patterns and highlight common errors. This phase ensures that every model aligns with the project's goals. If the performance is unsatisfactory, previous phases, including data preparation or model optimisation, would be revisited to implement needed enhancements.

3.2.3.6 Deployment

The top-performing model will be deployed into a basic web application using Streamlit during the deployment stage. The web application can take in users' debates or speech transcripts via a text input area. The system will process the text and render the argument types, identified fallacies, and basic visualisations. Based on these results, the system will send the analysis to activate an LLM model to generate personalised suggestions for enhancing the argument. This web interface makes the tool simple to operate and accessible to even non-experts. The deployment stage wraps

up the project by translating the trained model into a practical, interactive tool for real-world scenarios.

3.3 Summary

In summary, Chapter 3 explained the methodological framework chosen for the project and justified the selection of CRISP-DM as the most suitable approach. CRISP-DM provides a structured process and guides the project from understanding the problem, exploring and preparing the data, to modelling, evaluation, and deployment. Each phase plays a crucial role in ensuring the debate analyser can be developed in the right way to minimise errors and align with project objectives.

Chapter 4: Design and Implementation

4.1 Introduction

Chapter 4 presents the end-to-end design and implementation of the debate analyser, which starts from raw IBM Debater transcripts to trainable datasets and models ready for evaluation. It begins with data collection and consolidation, then validates and cleans the corpus before expanding it to the sentence level with punctuation restoration and targeted discourse-cue splitting for reliable segmentation. The chapter then imports an external fallacy detector for feasibility checks, samples a balanced subset of debates for manual Toulmin argument type labelling and uses a fine-tuned DeBERTa model to fill in these labels to the full corpus. With a complete sentence-level dataset, the text is prepared for multiple learners, including an SVM baseline, a TextCNN, and several transformer models like DeBERTa, GPT, and smaller QLoRA variants of DeepSeek and TinyLLaMA. All transformer models are trained under a consistent Trainer pipeline with accuracy and macro-F1 as primary metrics. Finally, this chapter will implement hyperparameter tuning, using a grid search for SVM and lightweight Optuna searches for CNN and transformers (on a 100-debate subset for speed).

4.2 Data Collection

The dataset used in this project is sourced from the IBM Project Debater. It was chosen for its high quality and completeness, as it provides clean, well-structured speech-to-text transcripts from real-world debates. This makes it more suitable than alternatives like DebateSum from Hugging Face (Roush & Balaji, 2020), which only includes extracted evidence and citations rather than full debate transcripts. An initial idea to scrape transcripts from YouTube was considered, but the quality of those transcripts was too low. They were unstructured and difficult to break into proper sentences. In contrast, the IBM dataset offers full debate content that is already well-organised and ready for further processing.

Dataset Source:
https://research.ibm.com/haifa/dept/vst/debating_data.shtml#Debate_Speech_Analysis

No.	Dataset	Reference	Speeches	Topics
1	IBM Debater® - Recorded Debating Dataset - Release #5 (Light version - no audio files) + Counter speech annotations	ACL 2020	3,562	440
2	IBM Debater® - Recorded Debating Dataset - Release #4 (Light version - no audio files) + Annotated general-purpose claim-rebuttal pairs	EMNLP 2019 EMNLP 2018	200	50
3	IBM Debater® - Recorded Debating Dataset - Release #3 (Light version - no audio files) + Annotated mined claims	ArgMining 2019 @ACL LREC 2018	400	200
4	IBM Debater® - Recorded Debating Dataset - Release #2 (Light version - no audio files) + Annotated arguments	EMNLP 2018 LREC 2018	200	50
5	IBM Debater® - Recorded Debating Dataset - Release #1 (Light version - no audio files)	LREC 2018	60	16

Table 6: IBM Debate Transcripts Download Sources

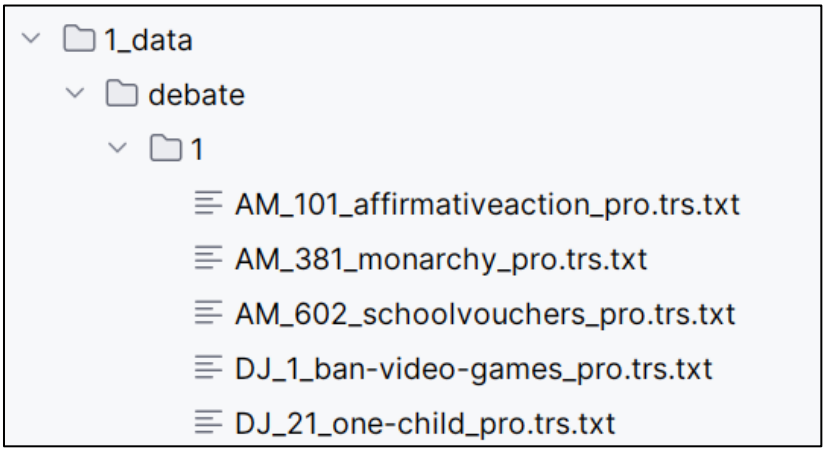


Figure 6: Debate Transcript Files Management

The debate transcripts were downloaded from the Debate Speech Analysis section, specifically selecting the "light" version of the transcripts. After downloading the .zip or .gz archive files, the text transcripts within the trs.txt folder were extracted. All the .txt files were moved into a newly created folder in the project to ensure better organisation and easier access for further processing.

4.3 Data Understanding and Data Pre-processing

Since each debate transcript is stored in a separate file at this stage, all text files are combined into a single CSV file for easier processing. Relevant information embedded in the filenames, such as stance and debate ID, is extracted and added as metadata columns. Once consolidated, exploratory data analysis (EDA) is performed to understand better the structure, distribution, and quality of the data. This includes examining sentence patterns, identifying inconsistencies, and uncovering potential issues that inform subsequent cleaning and preparation steps.

```
import os
import re
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.python.ops.gen_array_ops import upper_bound
from wordcloud import WordCloud
import nltk
from nltk.tokenize import word_tokenize
from nltk.tokenize import sent_tokenize
from nltk import pos_tag
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords, wordnet as wn
import torch
from deepmultilingualpunctuation import PunctuationModel
from sklearn.feature_extraction.text import CountVectorizer
from collections import Counter
from transformers import pipeline

nltk.download('punkt')
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger_eng')
nltk.download('wordnet')
```

Code Snippet 1: Import Libraries

Necessary libraries are imported for text processing, tokenisation, visualisation, feature extraction, and applying transformer-based models for NLP tasks.

4.3.1 Transcript Compilation and Extraction

```
# Folder containing raw transcripts
base_folder = "1_raw_data/debate"
all_data = []

for root, dirs, files in os.walk(base_folder):
    for file in files:
        if file.endswith(".txt"):
            file_path = os.path.join(root, file)
            try:
                with open(file_path, "r", encoding="utf-8") as f:
                    transcript = f.read().strip()

                    # Extract metadata from filename
                    match = re.match(r"([a-zA-Z]+)_(\d+)_([a-zA-
Z\-]+)_(pro|con)", file)
                    if match:
                        debater_name = match.group(1)
                        debate_id = match.group(2)
                        topic = match.group(3)
                        stance = match.group(4)
                    else:
                        debater_name = "unknown"
                        debate_id = "unknown"
                        topic = "unknown"
                        stance = "unknown"

                    # Count word length using nltk tokenizer
                    word_count = len(word_tokenize(transcript))

                    # Append to list
                    all_data.append({
                        "filename": file,
                        "debate_id": debate_id,
                        "debater_name": debater_name,
                        "topic": topic,
                        "stance": stance,
                        "transcript": transcript,
                        "transcript_length": word_count
                    })

            except Exception as e:
                print(f"Error reading {file_path}: {e}")

df = pd.DataFrame(all_data)
df.head()
```

Code Snippet 2: Transcript Compilation and Extraction

filename	debate_id	debater_name	topic	stance	transcript	transcript_length
AM_101_affirmativeaction_pro.trs.txt	101	AM	affirmativeaction	pro	The motion before us is this house would support affirmative action. The problem that we're presented with is that across society there continues to be insufficient representation and advancement of women in major roles, particularly in major companies, political positions and so forth. Therefore, one mechanism of curing this to advance women's roles to see have women act as leaders and to be included not only generally in the work force but	1342
AM_381_monarchy_pro.trs.txt	381	AM	monarchy	pro	In this speech I'm going to argue to you that this house should abolish the monarchy the problem that we're presented with its that the monarchy specifically in a number of western countries particularly on continental europe but notably in england and england will be talking about specifically continues to exist despite the fact that these countries are and and the UK in particular is is is a democratic society. Having it what is in effect a non elected official who gains their title on the basis of birth or in previous times a divine right is a a a is a inegalitarian, illegitimate	1261
AM_602_schoolvouchers_pro.trs.txt	602	AM	schoolvouchers	pro	The motion before us is that this house would create a system of school vouchers. School vouchers, fundamentally, are cash grants or subsidies to parents of children that they can then use for the purchase of education from any school which they desire. It involves getting rid of government provision or running of educational institutions and and pre-secondary and pre-college k eight through twelve schools but it	1346
DJ_121_ban-boxing_pro.trs.txt	121	DJ	ban-boxing	pro	We should ban boxing because boxing is a blood sport and it's absolutely horrible for the athletes and it's a horrible message to send for society. So why is this true? The first big thing is that boxing has horrific consequences on the people who participate in it. It's essentially like a modern day equivalent of gladiator battles in like the worse way possible.	689
DJ_1_ban-video-games_pro.trs.txt	1	DJ	ban-video-games	pro	We should ban the sale of violent video games to minors because it normalizes violence and it does so in a way that's particularly harmful because it does it to impressionable young people. So let's start by talking about how this normalizes violence. And I think that while it's probably unlikely that most of the time when someone picks up call of duty or grand theft auto or something like that, most people	592

Figure 7: Compilation Output

In this stage, all raw debate transcript files from the debate source folder are read and compiled into a single CSV file. Key metadata, such as debate ID, debater name, topic, and stance, are extracted from the filename using regular expressions for each file. The transcript content is tokenised using NLTK to calculate its word count, which is recorded as the transcript length. This process creates and saves a structured dataset in the processed dataset folder. This file serves as the foundation for subsequent stages, including exploratory data analysis (EDA), sentence segmentation, and argument labelling.

4.3.2 Transcript Dataset Validation

```
# Dataset Info
print("\033[94m\n--- Dataset Info ---\033[0m")
print(df.info())
```

Code Snippet 3: Print Dataset Info

```
RangeIndex: 4422 entries, 0 to 4421
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   filename           4422 non-null   object
 1   debate_id          4422 non-null   int64
 2   debater_name       4422 non-null   object
 3   topic              4422 non-null   object
 4   stance             4422 non-null   object
 5   transcript         4422 non-null   object
 6   transcript_length  4422 non-null   int64
dtypes: int64(2), object(5)
memory usage: 242.0+ KB
```

Figure 8: Dataset Info

The combined debate transcript dataset contains seven columns and 4,422 entries, each showing a complete set of non-null values, indicating no missing data. This confirms that all transcript files are successfully compiled and that the metadata extraction process is consistent. Understanding the dataset structure at this stage is crucial to ensure smooth execution in the following stages.

```
# Check for 'unknown' values in key metadata columns
print("\033[94m\n--- Count of 'unknown' in Key Columns ---
\033[0m")
for col in ['debate_id', 'debater_name', 'topic',
'transcript']:
    count = (df[col] == 'unknown').sum()
    print(f"{col}: {count} row(s) with 'unknown'")
```

Code Snippet 4: Count "unknown" Value

```
--- Count of 'unknown' in Key Columns ---
debate_id: 0 row(s) with 'unknown'
debater_name: 0 row(s) with 'unknown'
topic: 0 row(s) with 'unknown'
transcript: 0 row(s) with 'unknown'
```

Figure 9: "unknown" Value Count

To validate the accuracy of the metadata extraction process, the dataset is checked for any occurrences of the value "unknown" in key columns generated from the filename, which are debate_id, debater_name, topic, and stance. The results show zero "unknown" values in all these columns, confirming that the extraction logic functions correctly.

```
# Check for duplicate filenames
print("\033[94m\n--- Duplicate Check ---\033[0m")
duplicate_count = df.duplicated(subset='filename', keep=False).sum()
duplicate_transcript_count = df.duplicated(subset='transcript',
keep=False).sum()
print(f"Duplicate filenames found: {duplicate_count}")
print(f"Duplicate transcript found: {duplicate_transcript_count}")
```

Code Snippet 5: Count Duplicated Rows

```
--- Duplicate Check ---
Duplicate filenames found: 458
Duplicate transcript found: 931
```

Figure 10: Duplicated Rows Count

```
# Remove duplicates for clean analysis
df_unique = df.drop_duplicates(subset='filename', keep='first')
df_unique = df.drop_duplicates(subset='transcript', keep='first')
```

Code Snippet 6: Remove Duplicated Rows

To ensure data consistency, the dataset is checked for duplicate entries based on filenames and transcript content, as the transcripts are collected from five different source folders. A total of 458 duplicate filenames and 931 duplicate transcripts are identified. These duplicates are then removed to retain unique records.

4.3.3 Transcript Dataset Exploration

```
# Category distributions
print("\033[94m\n--- Unique Debate ID Counts ---\033[0m")
print(df_unique['debate_id'].value_counts())

print("\033[94m\n--- Unique Debater Name Counts ---\033[0m")
print(df_unique['debater_name'].value_counts())

print("\033[94m\n--- Unique Topic Counts ---\033[0m")
print(df_unique['topic'].value_counts())
```

Code Snippet 7: Print Category Distribution

```
--- Unique Debate ID Counts ---
debate_id
1161    25
3191    24
2480    24
381     24
2143    23
        ..
4003     1
4147     1
4354     1
4362     1
1246     1
Name: count, Length: 442, dtype: int64
```

Figure 11: Debate_id Category Count

```
--- Unique Debater Name Counts ---
debater_name
JL        768
TL        764
RG        688
SSH       475
WS        462
DJ        432
YBA       210
EH        101
HE         71
james      49
will       40
tim        30
rachel     26
yaar       20
daniel     20
sam         9
YB          6
SF          5
SN          5
hayah       4
DZ          3
AM          3
eitan       2
Name: count, dtype: int64
```

Figure 12: Debater_name Category Count

```
--- Unique Topic Counts ---
topic
racial-profiling           40
recall-elections           34
television                 29
goal-line                  26
stem-cell                  26
                           ..
minority-groups             1
sustainable-development     1
privatization               1
athletes                    1
planned-parenthood          1
Name: count, Length: 401, dtype: int64
```

Figure 13: Topic Count

The transcript dataset is further explored by analysing the distribution of key metadata. The debate_id reflects the session or round of a debate, with some IDs containing up to 25 related transcripts, suggesting multiple participants or segments per session in the dataset. For debater_name, identifiers like "JL" and "TL" appear frequently. However, these anonymised names provide limited interpretability and are likely placeholders or pseudonyms, offering minimal value for argumentation analysis. The topic field shows repeated entries, such as “racial-

profiling" and "recall-elections," which indicate different stances (pro/con) across participants. While this metadata is helpful for organisational purposes, it does not reveal meaningful patterns that directly contribute to the classification of argument types or the detection of logical fallacies.

```
# Stance distribution plot
plt.figure(figsize=(6, 4))
ax = sns.countplot(data=df_unique, x='stance', palette='Set2')
for bar in ax.patches:
    ax.annotate(f"{bar.get_height()}", (bar.get_x() + 0.3,
bar.get_height() + 5))
plt.title("Stance Distribution")
plt.xlabel("Stance")
plt.ylabel("Count")
plt.tight_layout()
plt.show()
```

Code Snippet 8: Create Stance Distribution Bar Chart

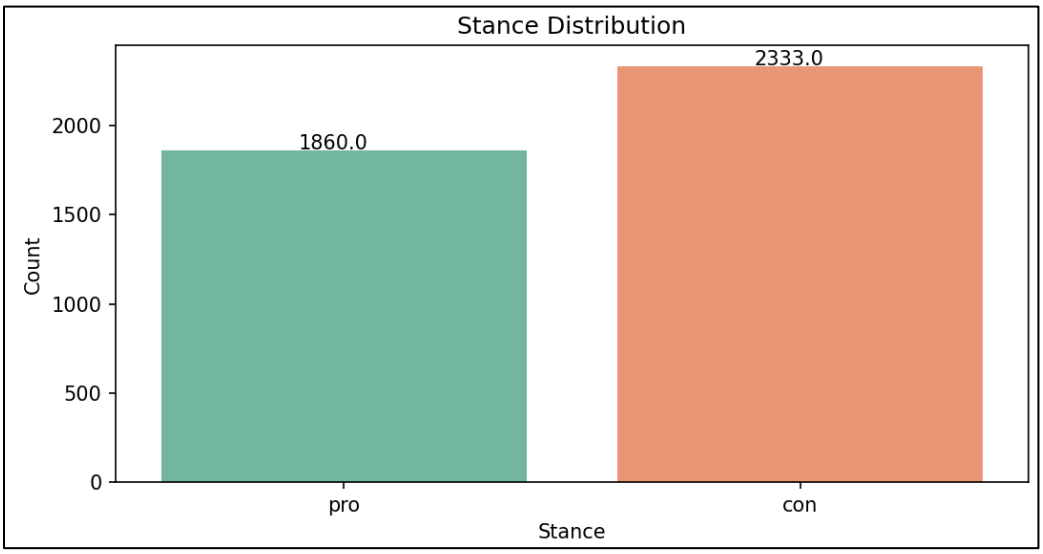


Figure 14: Stance Distribution

The dataset's stance distribution shows 1,860 transcripts labelled "pro" and 2,333 as "con." This column is retained to investigate whether debaters' stance influences their use of argument types or logical fallacies. Understanding this relationship could provide insights into how argumentative strategies differ between supporting and opposing positions.

```
# Word Cloud - All transcripts
text_all = " ".join(df_unique['transcript'].astype(str))
wordcloud_all = WordCloud(width=800, height=400,
background_color='white').generate(text_all)
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud_all, interpolation='bilinear')
plt.axis('off')
plt.title("Word Cloud - All Transcripts")
plt.tight_layout()
plt.show()
```

Code Snippet 9: Create Transcript Word Cloud

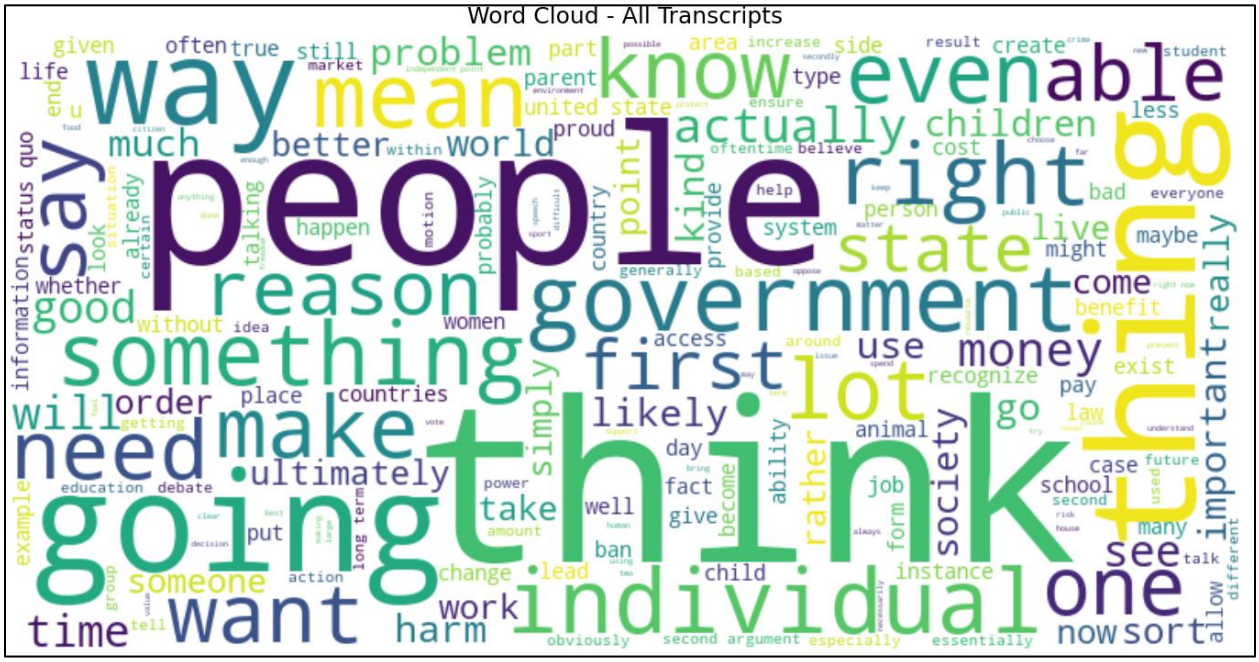


Figure 15: Transcript Word Cloud

The word cloud is used to identify the most frequently used words across all debate transcripts. It reveals that frequently occurring terms include "people," "think," "government," "individual," and "going." These prominent words reflect the inherently argumentative and persuasive nature of debates, where speakers often construct logical reasoning and present claims.

```
# Word Cloud - Pro vs Con
text_pro = " ".join(df_unique[df_unique['stance'] == 'pro']['transcript'])
text_con = " ".join(df_unique[df_unique['stance'] == 'con']['transcript'])

wordcloud_pro = WordCloud(width=800, height=400,
background_color='white').generate(text_pro)
wordcloud_con = WordCloud(width=800, height=400,
background_color='white').generate(text_con)

plt.figure(figsize=(10, 5))
plt.imshow(wordcloud_pro, interpolation='bilinear')
plt.axis('off')
plt.title("Word Cloud - Pro Stance")
plt.tight_layout()
plt.show()

plt.figure(figsize=(10, 5))
plt.imshow(wordcloud_con, interpolation='bilinear')
plt.axis('off')
plt.title("Word Cloud - Con Stance")
plt.tight_layout()
plt.show()
```

Code Snippet 10: Create Transcript Word Cloud by Stance

Figure 16: Transcript Word Cloud by Pro

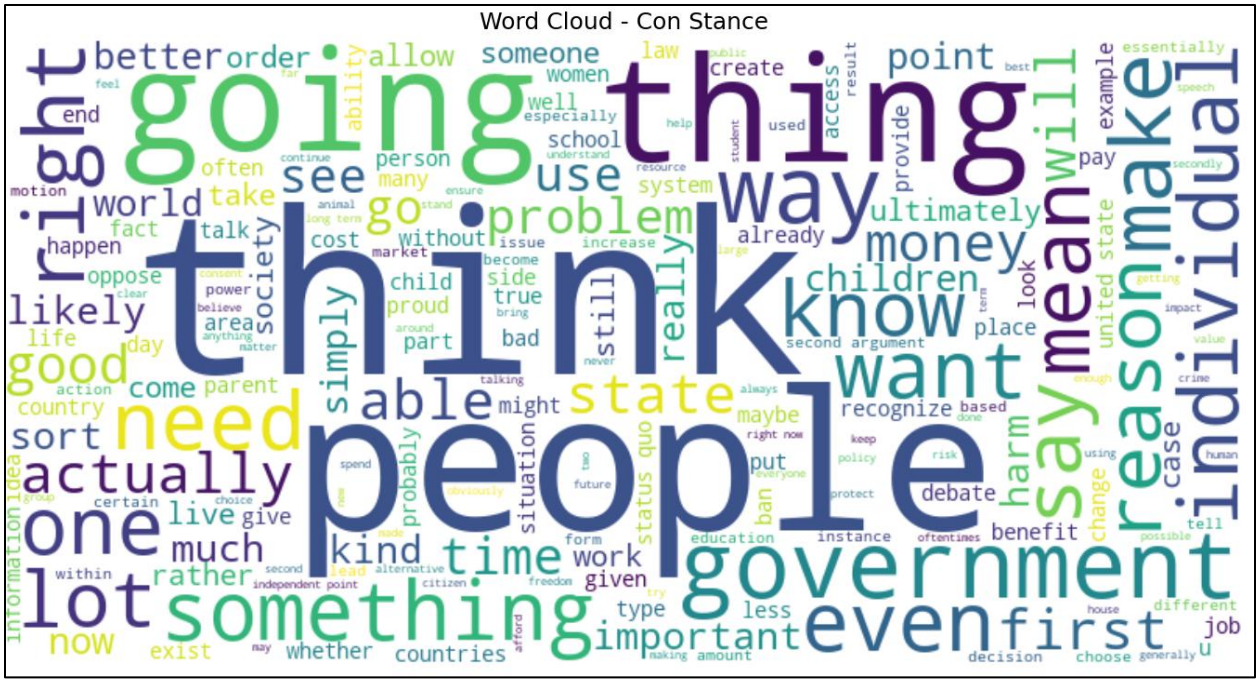


Figure 17: Transcript Word Cloud by Con

The word clouds compare the pros and cons of the vocabulary commonly used by debaters. Both word clouds show a substantial overlap in frequently used terms such as “think,” “people,” “thing,” “going,” and “government.” This overlap reflects the general structure and style of persuasive debate language.

```
# Histogram - Transcript Length
plt.figure(figsize=(8, 4))
sns.histplot(df_unique['transcript_length'], bins=30, kde=True,
color='skyblue')
plt.title("Transcript Length Distribution")
plt.xlabel("Transcript Word Count")
plt.ylabel("Number of Debates")
plt.tight_layout()
plt.show()
```

Code Snippet 11: Create Transcript_length Histogram

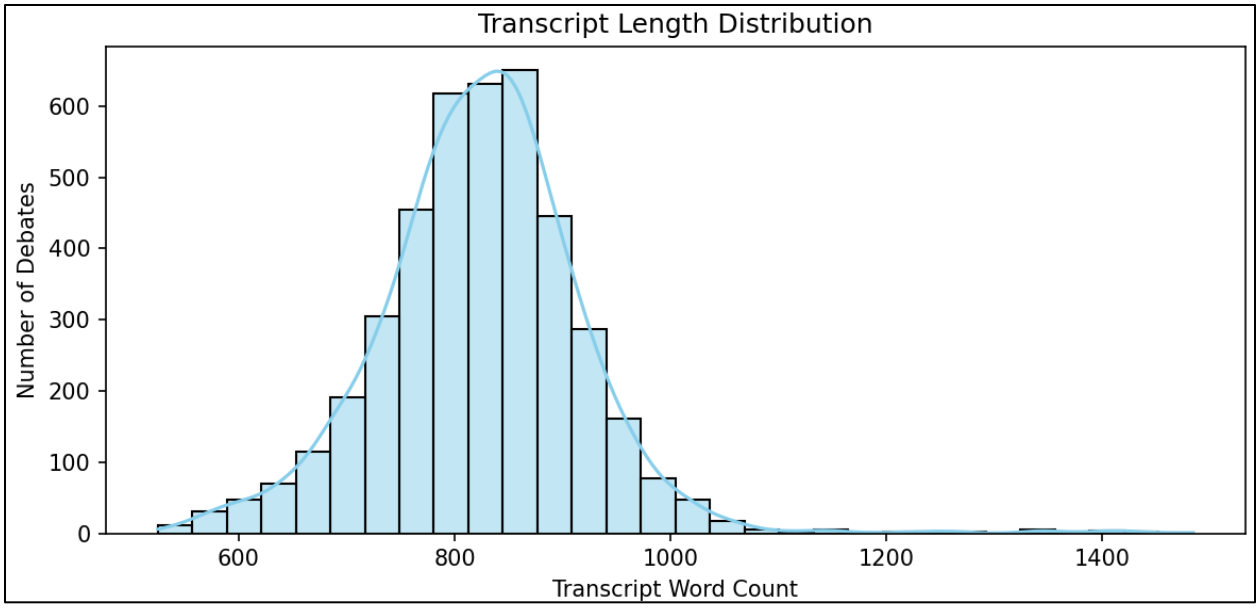


Figure 18: Transcript_length Distribution

The histogram shows the distribution of transcript lengths based on word count across the debate dataset. The distribution looks normal, with most transcripts ranging between 750 and 900 words, and a clear peak around 850 words. A few debates exceed 1,200 words, contributing to a slight right skew.

```
# Boxplot - Transcript Length
plt.figure(figsize=(6, 4))
sns.boxplot(y=df_unique['transcript_length'],
color='lightcoral')
plt.title("Transcript Length Boxplot")
plt.ylabel("Word Count")
plt.tight_layout()
plt.show()
```

Code Snippet 12: Create Transcript_length Boxplot

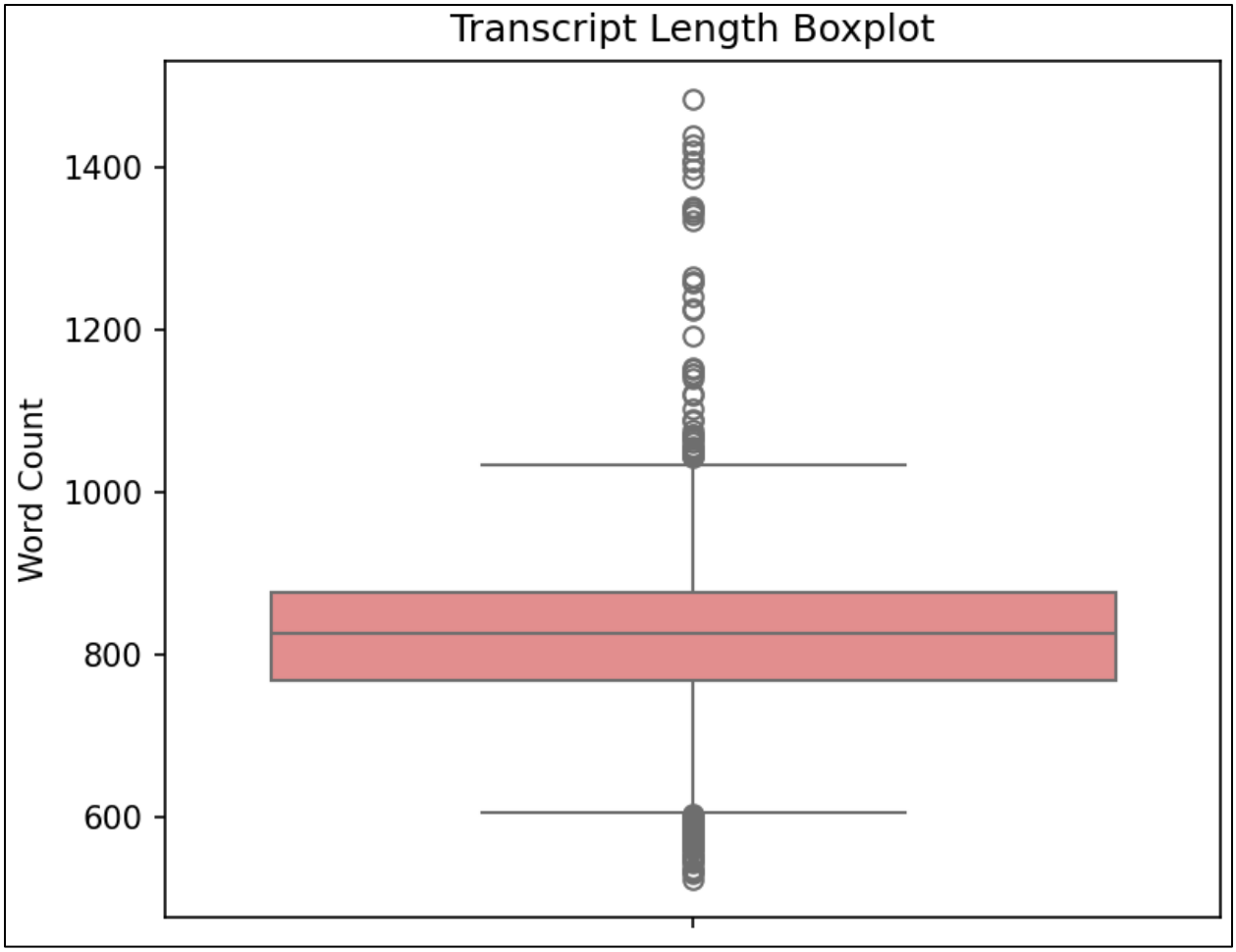


Figure 19: Transcript_length Boxplot

The boxplot shows the distribution and spreads of transcript lengths. Most transcripts fall within the interquartile range of approximately 770 to 880 words, with a median centred around 825. Several outliers are present, exceeding 1,050 words, and others drop below 600. These outliers may reflect differences in speaking style, pacing, or debate structure.

4.3.4 Outliers and Weak Features Removal

```
# Calculate IQR to detect outliers in transcript_length
Q1 = df_unique['transcript_length'].quantile(0.25)
Q3 = df_unique['transcript_length'].quantile(0.75)
IQR = Q3 - Q1

# Define upper and lower bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter out outliers
df_filtered = df_unique[(df_unique['transcript_length'] >=
lower_bound) & (df_unique['transcript_length'] <= upper_bound)]
```

Code Snippet 13: Remove Outliers

Outliers in transcript length are removed to ensure more consistent and reliable data for modelling. Since debate transcripts are expected to follow a relatively uniform structure, unusually short transcripts may lack sufficient argument components. In contrast, overly long ones can introduce noise or exceed token limits in transformer-based models, leading to inefficiencies in training and inference. The interquartile range (IQR) method identifies and filters out transcripts outside the typical word count range.

```
# Drop weak columns
df_drop = df_filtered.drop(["debate_id", "debater_name", "topic"], axis=1)
df = df_drop

df.head()
```

Code Snippet 14: Remove Weak Columns

Columns such as debate_id, debater_name, and topic are removed because they do not contribute meaningful features for argument classification or fallacy detection. Dropping these non-informative columns helps simplify the dataset.

filename	stance	transcript	transcript_length
DJ_121_ban-boxing_pro.trs.txt	pro	We should ban boxing because boxing is a blood sport and it's absolutely horrible for the athletes and it's a horrible message	689
DJ_21_one-child_pro.trs.txt	pro	The one child policy brought more good than harm to china, because it was an act of necessity, and while it may not have	947
DJ_482_tobacco_pro.trs.txt	pro	We should absolutely not subsidize the cultivation of tobacco because tobacco is bad and you shouldn't subsidize things	782
DJ_644_year-round-schooling_pro.trs.txt	pro	Year round schooling will bring more good than harm for two big reasons.	843
DJ_681_ip-rights_pro.trs.txt	pro	We should abolish intellectual property rights because intellectual property rights do a tremendous amount of social	792

Figure 20: Processed Transcript Dataset

The processed transcript dataset is saved as a checkpoint to preserve a clean and standardised data version. This enables easy rollback if adjustments are needed later and ensures the dataset is well-prepared for the next stage of the workflow.

4.3.5 Dataset Expansion to Sentence-level

```
# Use nltk to initially split transcripts into sentences
sentence_data = []
for _, row in df.iterrows():
    transcript = str(row["transcript"]).strip()
    nltk_sentences = sent_tokenize(transcript)

    for i, sentence in enumerate(nltk_sentences, start=1):
        sentence_data.append({
            "filename": row["filename"],
            "stance": row["stance"],
            "sentence": sentence.strip(),
            "sentence_order": i,
            "sentence_length": len(sentence.strip().split())
        })

sentence_df = pd.DataFrame(sentence_data)
```

Code Snippet 15: Break Transcript into Sentence-Level

The transcript data is expanded into individual sentences using NLTK's sent_tokenize function to facilitate sentence-level analysis. For each sentence, the script captures its original filename, stance, position within the transcript (sentence_order), and the number of words (sentence_length).

```
# Plot histogram of sentence lengths
plt.figure(figsize=(8, 4))
plt.hist(sentence_df["sentence_length"], bins=100, color='skyblue',
edgecolor='black')
plt.title("Sentence Length Distribution")
plt.xlabel("Number of Words per Sentence")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()

# Plot violin plot of Sentence Length
plt.figure(figsize=(6, 4))
sns.violinplot(y=sentence_df['sentence_length'], color='lightcoral')
plt.title("Sentence Length Violin Plot")
plt.ylabel("Word Count")
plt.tight_layout()
plt.show()
```

Code Snippet 16: Create Sentence Length Histogram and Violin Plot

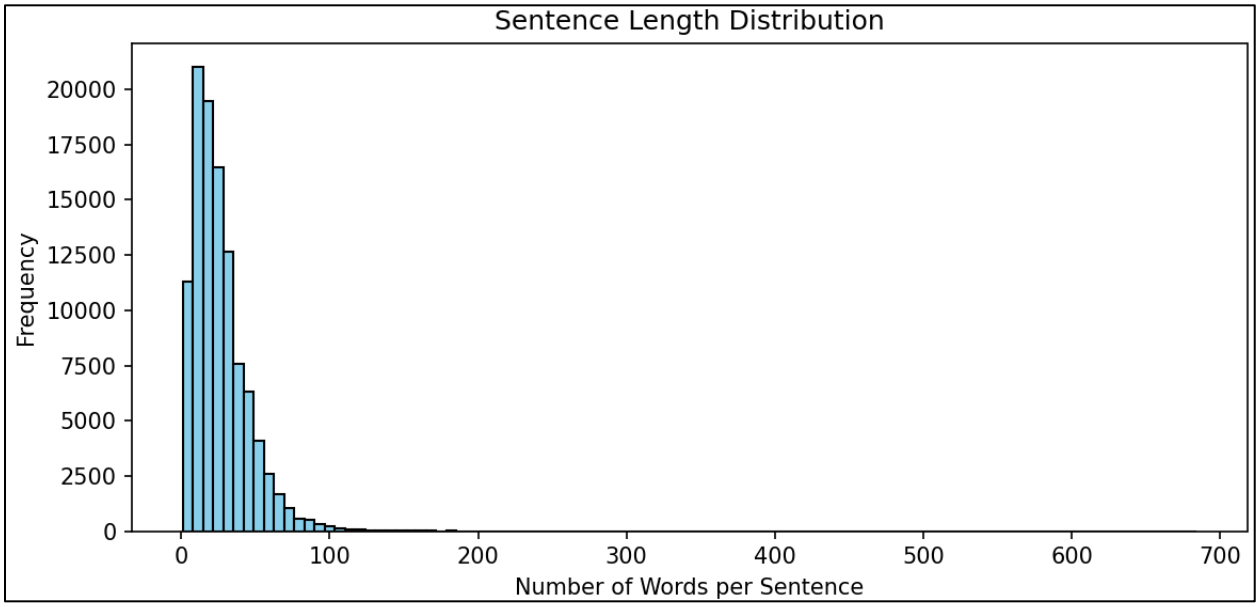


Figure 21: Sentence Length Distribution

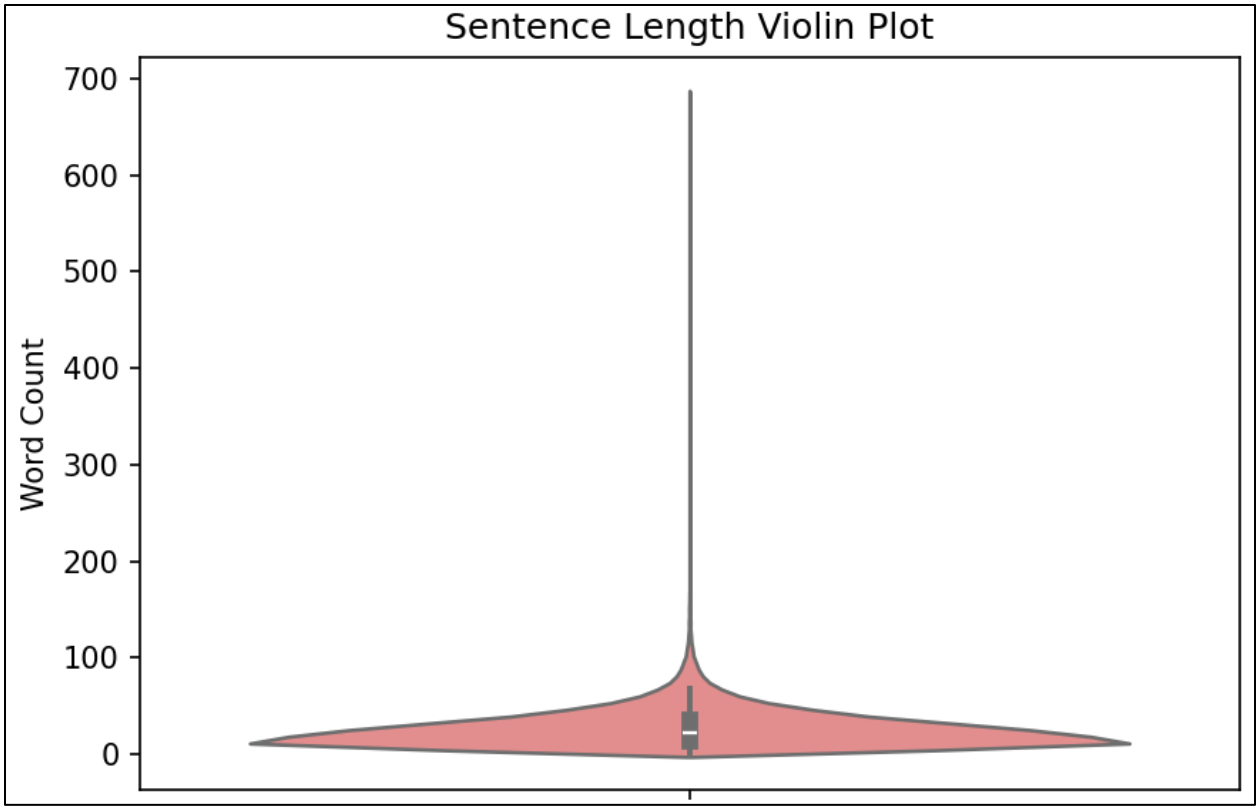


Figure 22: Sentence Length Violin Plot

The sentence length histogram shows a heavily left-skewed distribution, indicating that most sentences are short, typically under 40 words, while a few sentences extend beyond 100 words. This long-tail distribution is also evident in the violin plot, highlighting several outliers with exceptionally high word counts. While these longer sentences may contain coherent content, they could also represent run-on sentences, multiple ideas merged, or formatting issues during transcript processing. Before deciding on cleaning or truncation actions, these long sentences should be manually reviewed for context to ensure important argument components are not unintentionally removed or distorted.

```
# Sort sentence_length to find longest and shortest
longest = sentence_df.sort_values(by="sentence_length",
ascending=False).head(2)
shortest = sentence_df.sort_values(by="sentence_length",
ascending=True).head(5)

# Print outliers results
for idx, row in longest.iterrows():
    print(f"\nFilename: {row['filename']}\n"
          f"Sentence Length: {row['sentence_length']} words\n"
          f"Sentence: {row['sentence']}")

for idx, row in shortest.iterrows():
    print(f"\nFilename: {row['filename']}\n"
          f"Sentence Length: {row['sentence_length']} word(s)\n"
          f"Sentence: {row['sentence']}")
```

Code Snippet 17: Print Two Longest and Five Shortest Sentences' Length Value

```
Filename: SSH_3232_pride-parades_con_opening_RG.trs.txt
Sentence Length: 684 words
Sentence: I stand in firm opposition to the motion we should cancel pride parades
pride parades are an essential tool for individuals who are not in the mainstream
in terms of gender sexuality to come together in solidarity and empower one another
ultimately pride parades are the decisions of the people who have faced this sort
of expression and as such the rights should not be abridged to that and I have
three arguments but first some brief about all my opponent spends most of their
speech focusing on the fact that pride parades are a target for hateful groups they
say that if we allow this target to exist then violence will occur because people
will act out against them so to responses first there are hundreds of targets
around the world around it honestly even just the state of massachusetts which are
worse than pride parades pride parades are a terrible spot for a terror or heat
attack because individuals can run away in numerous directions they're not in an
enclosed area and they usually have police officers either attending or around them
and there are hundreds of witnesses there are other targets like queer clubs like
support groups like focus group that all focus on LGB teach you individuals we saw
this in paris where hundreds of people were killed I'm at a club by terrorists
because they want to take action against a certain group we also saw in florida at
that queer individuals are targets wherever they go we think that pride parades are
actually on the safer ways in which we are people can express themselves and as
such my points arguments just do not stand the second we all think this is the real
reason to stop a particular action white supremacists have targeted black churches
since the inception of america and the introduction of christianity to black people
this is not a reason to shut down every black church would fundamentally a bridge
the rights of individuals to use the oppressors of violence against them to take
away their rights and take away their activities please when I first argument LGB
teach you individual if visibility we think that individuals are far more likely to
come out if they feel like there's a community which will accept them additionally
it individuals will feel more comfortable forming social groups I'm going on the
public if they know that other people are also queer are also I'm a questioning
individual and we think that's what the visibility empowers people it empowers
people who are already out but for people who are still in the closet it tells them
that they can come out live a good life and have a culture which is unique to them
and which they have access to now they want to come out my second argument is LGB
teach you solidarity we think that pride parades have a long history which goes
back to protests against the violence against all the BTU individuals both
individuals and by police officers this sort of solidarity is the best place for
these protests for protest to occur it allows individuals to express their views to
talk about how police continue to fail to protect LGB DQ individuals this sort of
solidarity creates a stronger political and economic bloc fork where individuals
which is the only way in which powerful organizations will respond to them they
meaningful way my third and final argument is it leads to greater discourse by
opponent argues that pride parades ultimately make people more hateful simply
because people will see more deviant expressions of sexuality and have backlash
against them whoever we think that individuals are far more likely to be engage in
discourse in this modern day where individuals are actually more enlightened or
they understand more about LGB DQ culture and the positions which they take
ultimately we think that this improves individuals understanding of queer issues
while also allowing people to have fun and act more effectively as a group for this
reason we should not cancel pride parades whole
```

Figure 23: Longest Sentence Length Value

```
Filename: SSH_3227_capital-punishment_con_RG_implicit.trs.txt
Sentence Length: 660 words
Sentence: I stand in firm opposition to the motion we should abolish capital
punishment all that capital punishment is not usually the best tool for dealing
with certain individuals it is essential to have it as a tool in the toolbox even
if it is never used the threat of capital punishment alone concur go to great
lengths in preventing further harms against civilians to that end I have three
arguments first that capital punishment is a tool in the toolbox recognize that
prosecutors judges juries all have discretion in deciding whether or not to give
someone capital punishment we think that individuals particularly those they strong
sense of justice are far less likely to use capital punishment an unjust ways what
we mean by this we think that prosecutors who went into law in the first place to
pursue justice and decide to work for the state rather than going to work for a
cushy and what well paying private job usually have a strong sense of justice they
want to fight for what is right in our society and what is best as such they're
unlikely to pursue charges that will ultimately lead to the death penalty judges
are individuals who are usually unable to be disbarred at this point unless they
actively fight against the good of the people and ultimately do something illegal
to that end because of the sort of job security and because of the position they
hold with our society we think that judges are unlikely to pursue capital
punishment either unless someone is a truly reprehensible individual finally we
think that juries usually have a decent of information they not only have the
information that comes from living in our society understanding of justice
statistics and understanding our laws but are also able to sit through a multiple
day trial in which to train legal professionals educate them about what they ought
to do and how our system works ultimately we have a number of checks and balances
to guarantee that innocent people do not get the death penalty that only that at
the death penalty becomes more unpopular we think that the death penalty is not a
political benefit either as such by continuing to have as a tool in the toolbox we
can punish the truly awful people the jeffrey dahmer's thought ted bundy's the
serial killers who have no remorse and ultimately torture the people who they
attempt to harm my second argument that this deters crime it's very difficult to
understand how bad prison is until you go there for many individuals this makes it
said they are unlikely to use present laws and premise present punishments as a
strong deterrent against criminal action we think that every individual understands
death at least to an extent which allows them to know they don't want to die
individuals are unlikely to pursue crime than given that adam they don't want to
decide they don't want to face the punishments of their actions and so they are
unlikely to take the most heinous action against our society by deterring crime
that we're doing the best possible good for innocent individuals my third and final
argument is that justice is idiosyncratic our government has put capital punishment
in the place anti capital punishment politicians have not taken over government
have not won many positions and as such it is clear that the people want capital
punishment it's impossible to derive any system of judges just punishment law from
any group besides the people we can't give it to any individual god because
discriminatory we can't give it any individual culture or nature because there's
simply no hierarchy from which we can derive the sort of considerations of justice
however by allowing the people to speak by long their preferences to guide our
systems of justice we can truly benefit the most people overall for this reason we
should not abandon capital punishment
```

Figure 24: Second Longest Sentence Length Value

```
Sentence Length: 1 word(s)
Sentence: Center.

Sentence Length: 1 word(s)
Sentence: Okay.

Sentence Length: 1 word(s)
Sentence: Right?

Sentence Length: 1 word(s)
Sentence: Right?

Sentence Length: 1 word(s)
Sentence: Look.
```

Figure 25: Top Five Shortest Sentence Length Value

After examining the sentence length distribution, it is found that extremely long sentences are mainly caused by inaccurate sentence segmentation due to missing punctuation in the original transcripts. These long segments often bundle multiple ideas into one, making them challenging for NLTK to analyse correctly. Manually segmenting in such cases can be time-consuming. Since the dataset consists of spoken debate transcripts, regular expressions are unsuitable, as they often produce incorrect or weak segmentations due to the irregular structure of speech. In this context, segmentation accuracy takes priority over speed, as poor sentence boundaries directly impact the quality of the next stage modelling. The researcher tries various methods, including spaCy, clause chunking, and the T5-based Benepar model. However, these approaches either fail to separate content without punctuation or split text solely based on length, which does not capture accurate sentence boundaries.

To address this, the solution involves first restoring punctuation using a pre-trained model. While ChatGPT provides highly accurate results for sentence segmentation, it is not feasible for large-scale processing due to the size of the dataset. Instead, a well-trained transformer-based model (*Oliverguhr/Fullstop-punctuation-multilang-large · Hugging Face*, 2022) is used to insert punctuation effectively. Once punctuation is restored, NLTK's sent_tokenize function is reapplied to split the text into individual sentences reliably. Very short sentences (fewer than five words) are filtered out unless they contain argument-related keywords such as "think," "should," or "believe," to ensure the dataset maintains high relevance and avoids filler or rhetorical fragments.

```
# IQR to identify long sentences
Q1 = sentence_df['sentence_length'].quantile(0.25)
Q3 = sentence_df['sentence_length'].quantile(0.75)
IQR = Q3 - Q1
upper_bound = Q3 + 1.5 * IQR

# Restore punctuation for long sentences
model = PunctuationModel()
keywords = ["should", "because", "think", "believe", "must", "argue"]
processed_sentences = []

for _, row in sentence_df.iterrows():
    sentence = row['sentence']
    sentence_length = row['sentence_length']

    if sentence_length > upper_bound:
        # Restore punctuation and segment again
        try:
            restored = model.restore_punctuation(sentence)
            split_sentences = sent_tokenize(restored)
        except Exception as e:
            print(f"Error restoring punctuation for: {sentence[:50]}... | {e}")
            split_sentences = [sentence]
    else:
        split_sentences = [sentence]

    for s in split_sentences:
        s = s.strip()
        word_count = len(s.split())
        if word_count < 5 and not any(k in s.lower() for k in keywords):
            continue
        processed_sentences.append({
            "filename": row["filename"],
            "stance": row["stance"],
            "sentence": s,
            "sentence_order": -1,  # placeholder
            "sentence_length": word_count
        })

# Assign sentence order per filename
sentence_df = pd.DataFrame(processed_sentences)
sentence_df["sentence_order"] = sentence_df.groupby("filename").cumcount() + 1
```

Code Snippet 18: Handle Meaningless Short and Overlong Sentences

The code first identifies long sentences using the interquartile range (IQR) method. Sentences that exceed the upper bound are flagged as overly long, often due to missing punctuation in the original transcripts. These are reprocessed using a transformer-based punctuation restoration model (oliverguhr/fullstop-punctuation-multilang-large) to recover sentence boundaries. The restored output is then segmented into individual sentences using NLTK's sent_tokenize function.

This selective application of the punctuation model ensures a balance between processing efficiency and segmentation accuracy. It avoids running the model on the entire dataset, which

would be computationally expensive, and instead focuses only on problematic sentences. Short sentences containing fewer than five words are excluded unless they include argument-relevant keywords such as “should,” “think,” or “believe,” ensuring that only contextually relevant content is retained. After segmentation and filtering, sentence order is recalculated for each transcript to maintain proper sequence.

```
# Print the processed sentences for the problem transcript
target_file = "SSH_3232_pride-parades_con_opening_RG.trs.txt"
subset = sentence_df[sentence_df["filename"] ==
target_file].sort_values("sentence_order")

print("Order: Content")
for _, row in subset.iterrows():
    print(f"{row['sentence_order']}: {row['sentence']}")
```

Code Snippet 19: Print Processed Sentences for Problem Transcript

```
Order: Content
1: I stand in firm opposition to the motion.
2: we should cancel pride parades.
3: pride parades are an essential tool for individuals who are not in the
mainstream in terms of gender sexuality to come together in solidarity and empower
one another.
...
...
...
32: we think that individuals are far more likely to be engage in discourse in
this modern day, where individuals are actually more enlightened or they
understand more about LGB DQ culture and the positions which they take.
33: ultimately, we think that this improves individuals understanding of queer
issues while also allowing people to have fun and act more effectively as a group.
34: for this reason we should not cancel pride parades whole.
```

Figure 26: Processed Sentences

After processing, the previously problematic transcript (SSH_3232_pride-parades_con_opening_RG.trs.txt) is reprinted with clearly segmented sentences. The issue of missing punctuation in long text is resolved.

```
# Plot histogram of sentence lengths
plt.figure(figsize=(6, 4))
sns.violinplot(y=sentence_df['sentence_length'], color='lightcoral')
plt.title("Sentence Length Violin Plot (After Processing with spaCy)")
plt.ylabel("Word Count")
plt.tight_layout()
plt.show()

# Plot violin plot of Sentence Length
plt.figure(figsize=(8, 4))
plt.hist(sentence_df["sentence_length"], bins=50, color='skyblue',
edgecolor='black')
plt.title("Sentence Length Distribution")
plt.xlabel("Number of Words per Sentence (After Processing with spaCy)")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```

Code Snippet 20: Create Histogram and Violin Plot After Processing

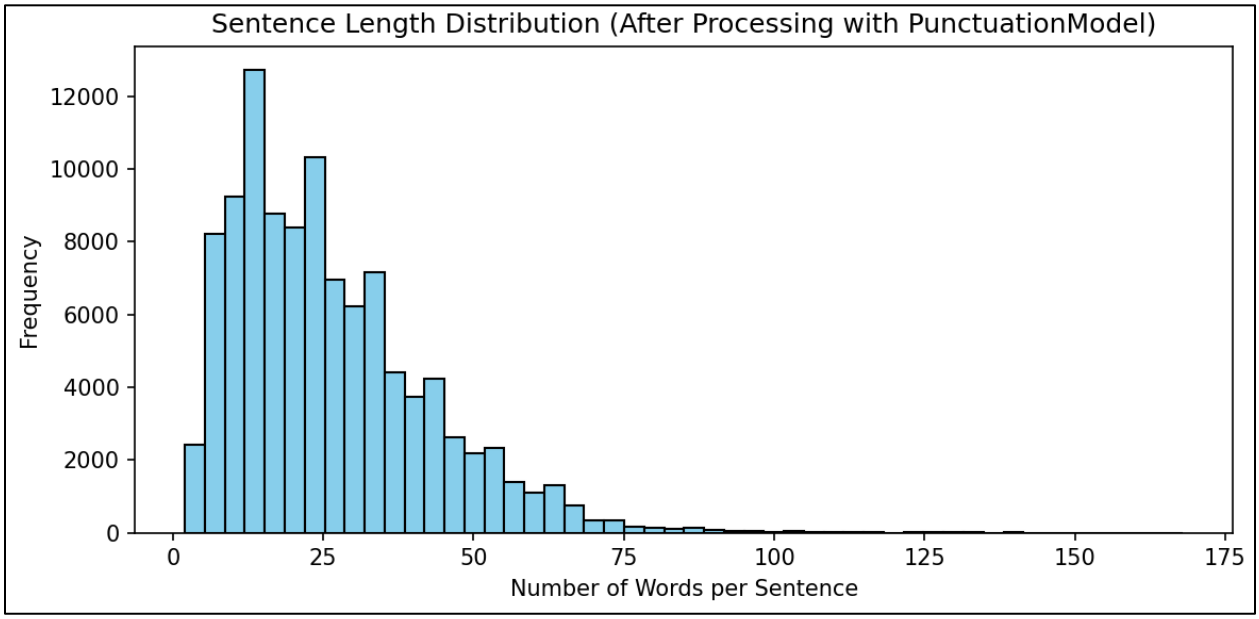


Figure 27: Sentence Length Distribution (After Spacy)

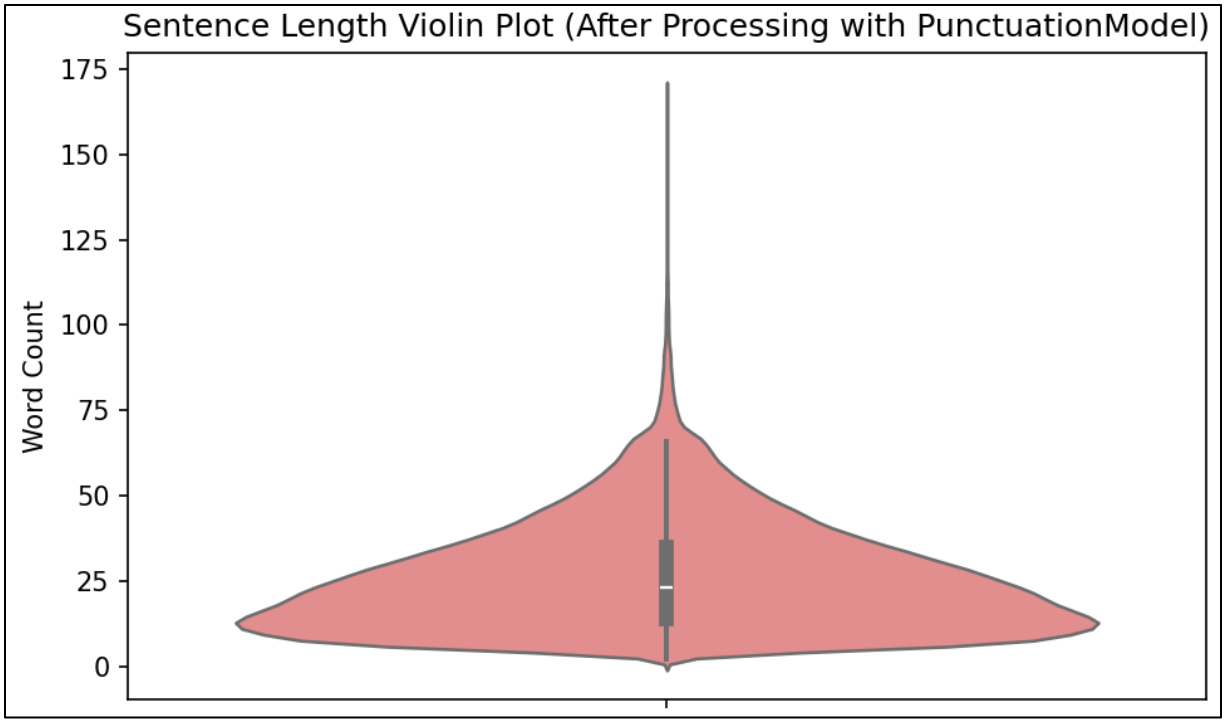


Figure 28: Sentence Length Violin Plot (After Spacy)

After processing, the sentence segmentation results improve, as reflected in the distribution and violin plots. However, a small number of sentences still exceed 100 words. While this may be acceptable in cases where debaters provide extended reasoning, it warrants a closer inspection to ensure these long segments are semantically valid and not the result of missed boundaries.

```
# Sort sentence_length to find longest and shortest
longest = sentence_df.sort_values(by="sentence_length",
ascending=False).head(1)

# Loop through the longest sentences and show the raw transcript for each
for idx, row in longest.iterrows():
    filename = row['filename']
    print(f"\nFilename: {filename}")
    print(f"Sentence Length: {row['sentence_length']} words")
    print("Processed Sentence:")
    print(row['sentence'])

    # Get the original transcript for comparison
    original_row = df[df['filename'] == filename]
    if not original_row.empty:
        raw_transcript = original_row.iloc[0]["transcript"]
        print("\nOriginal Transcript (Before Processing):")
        print(raw_transcript.strip())
    else:
        print("\nOriginal transcript not found.")
```

Code Snippet 21: Print Longest Processed and Original Sentence

```
Filename: DJ_3238_palestinian-independence_con_TL_implicit.trs.txt
Sentence Length: 168 words
Processed Sentence:
There's only limited political autonomy for the actual people, and the
reason for that is because they're worried about terrorism or even more
radical groups taking over, and I think that the clearest examples of this
are in gaza, where, after the israelis withdrew in the early two thousands
out of gaza out of hopes that this would help spark peace, renewed peace
agreements and viability of a palestinian state, the result is that fair
elections, election- well, no elections were held and radical groups like
hamas were able to leverage antisemitism, sectional fears and also scare
tactics like threatening opponents and using violence and intimidation-
voter intimidation and also candidate intimidation- as a means by which to
essentially monopolize the levers of government in gaza, and ultimately,
that led to a huge destabilization region where it became much more poor,
much more less economically viable, much, much more dangerous in terms of,
like the military groups that are active there and terrorist organizations
that are active there.

Original Transcript (Before Processing):
...
...
There's only limited political autonomy for the actual people and the
reason for that is because they're worried about terrorism or even more
radical groups taking over and I think that the clearest examples of this
are in gaza where after the israelis withdrew in the early two thousands
out of gaza out of hopes that this would help spark peace renewed peace
agreements and viability of a palestinian state the result is that fair
elections election well no elections were held and radical groups like
hamas were able to leverage antisemitism sectional fears and also scare
tactics like threatening opponents and using violence and intimidation
voter intimidation and also candidate intimidation as a means by which to
essentially monopolize the levers of government in gaza and ultimately
that led to a huge destabilization region where it became much more poor,
much more less economically viable, much much more dangerous in terms of
like the military groups that are active there and terrorist organizations
that are active there rockets are fired out of gaza into israel on a
regular basis all of those things would likely only be amplified if now
they had total independence where you can't have blockades on independent
state because that would be an act of war we're not going to have any kind
of reasonable security oversight of ...
...
...
```

Figure 29: Longest Sentence Length Value After and Before Processed

The remaining long sentences in the processed dataset show that the punctuation restoration model is inaccurate in separating ideas into proper sentences. In some cases, it inserts commas instead of full stops, particularly when debaters continue speaking using words like “and,” “but,” or “so.” As a result, these sentences remain unbroken, and nltk.sent_tokenize() cannot segment them correctly. To further address this issue, the saved processed dataset is re-examined. The final solution is manually reviewing the 20 longest processed sentences to identify recurring linguistic patterns that signal natural sentence boundaries.

Phrase	Why split using this phrase	Example
we think that	It starts a new argument or opinion from the speaker.	"We think that pride parades... We think that in many cases..."
and we think that	It continues the flow but starts a new point or justification.	"...and we think that this puts pressure..."
this means that	It introduces a consequence or an explanatory remark.	"...and this means that the reactors are less safe..."
the reason for this is that	It explains the cause behind the previously stated idea.	"The reason for this is that water access is limited..."
what this means is that	It clarifies the implication of the previous statement.	"What this means is that preachers are not priests..."
this is why	It introduces a justification or conclusion.	"This is why green tech helps economies..."
finally / the final thing is	It signals the final point or a shift toward conclusion.	"Finally, we want to talk about the insecurity of telemarketing..."
first, second, third	It introduces ordered reasoning or step-by-step arguments.	"First, this affects security... Second, it weakens trust..."
one, two, three	It introduces listed consequences or reasons.	"...and one, it puts pressure... and two, it stresses people..."
but	It introduces a contrast or counterpoint to the previous idea.	"...but we think the problem with this is..."
so	It presents a conclusion or result based on prior reasoning.	"...so this creates bad hiring practices..."
because	It introduces a cause or justification for the preceding claim.	"...because society stereotypes women as..."

Table 7: Discourse Cues for Sentence Splitting

```
# Define discourse-based splitter
def discourse_split(text):
    text = re.sub(r'\s+', ' ', text).strip()
    cues = [
        r'\band we think that\b',
        r'\bwe think that\b',
        r'\bthe reason for this is that\b',
        r'\bthis means that\b',
        r'\bwhat this means is that\b',
        r'\bthis is why\b',
        r'\bfinally\b',
        r'\bfirst\b', r'\bsecond\b', r'\bthird\b',
        r'\bone\b', r'\btwo\b', r'\bthree\b',
        r'\bbut\b', r'\bso\b', r'\bbecause\b'
    ]
    for cue in cues:
        text = re.sub(cue, r'. \g<0>', text, flags=re.IGNORECASE)
    return [s.strip() for s in re.split(r'\.\s*', text) if
len(s.strip().split()) >= 5]

# Process
new_sentences = []

for _, row in sentence_df.iterrows():
    sentence = row['sentence']
    length = row['sentence_length']

    # Only apply splitting if sentence exceeds upper bound
    if length > upper_bound:
        split_sentences = discourse_split(sentence)
    else:
        split_sentences = [sentence]

    for s in split_sentences:
        s = s.strip()
        word_count = len(s.split())
        if word_count < 5 and not any(k in s.lower() for k in keywords):
            continue
        new_sentences.append({
            "filename": row["filename"],
            "stance": row["stance"],
            "sentence": s,
            "sentence_length": word_count,
            "sentence_order": -1  # Will be reassigned
        })

# Reassign sentence order
final_df = pd.DataFrame(new_sentences)
final_df["sentence_order"] = final_df.groupby("filename").cumcount() + 1
```

Code Snippet 22: Split Sentences based on Discourse Cues

This code improves sentence segmentation in the debate dataset by applying a custom rule-based splitter to overly long sentences. It first defines a discourse_split() function that detects natural sentence boundaries using familiar discourse cues and inserts full stops before them. During processing, only sentences exceeding the IQR-based upper length threshold are split using this

function, while others remain unchanged. After splitting, the sentence order is reassigned within each transcript file to maintain sequential structure.

```
# Stats summary after processing
min_len = final_df["sentence_length"].min()
max_len = final_df["sentence_length"].max()
avg_len = final_df["sentence_length"].mean()
Q1 = final_df["sentence_length"].quantile(0.25)
Q3 = final_df["sentence_length"].quantile(0.75)
IQR = Q3 - Q1

print("\nSentence Length Statistics (After Splitting):")
print(f"Min: {min_len} words")
print(f"Max: {max_len} words")
print(f"Average: {avg_len:.2f} words")
print(f"IQR: {IQR} (Q1: {Q1}, Q3: {Q3})")
```

Code Snippet 23: Print Sentence Length Summary After Processed

```
Sentence Length Statistics (After Splitting):
Min: 2 words
Max: 127 words
Average: 25.48 words
IQR: 22.0 (Q1: 13.0, Q3: 35.0)
Upper Bound (Q3 + 1.5*IQR): 68.0
```

Figure 30: Sentence Length Summary

filename	stance	sentence	sentence_length	sentence_order
DJ_121_ban-boxing_pro.trs.txt	pro	We should ban boxing because boxing is a blood sport and it's absolutely horrible for the athletes and it's a horrible message to send for society.	26	1
DJ_121_ban-boxing_pro.trs.txt	pro	So why is this true?	5	2
DJ_121_ban-boxing_pro.trs.txt	pro	The first big thing is that boxing has horrific consequences on the people who participate in it.	17	3
DJ_121_ban-boxing_pro.trs.txt	pro	It's essentially like a modern day equivalent of gladiator battles in like the worse way possible.	16	4
DJ_121_ban-boxing_pro.trs.txt	pro	bones breaking in people's face, someone's eyes can get like gouged out and damage from like from like hits.	32	5
DJ_121_ban-boxing_pro.trs.txt	pro	There's all kinds of horrific injuries that people can obtain in the short term.	14	6
DJ_121_ban-boxing_pro.trs.txt	pro	A lot of those injuries specifically can also of long term impacts.	12	7

Figure 31: Sentence Dataset Output

After applying the custom sentence splitting and filtering logic, the sentence length distribution significantly improves, with most sentences falling within a manageable range. While a few long sentences remain (e.g., up to 127 words), the upper bound for typical sentence length is 68, and

the average is around 25. Given the nature of spoken debate transcripts, where speakers often elaborate at length or deliver extended reasoning, it is reasonable to expect some longer sentences. Since most problematic cases have been addressed and the remaining outliers do not appear to distort the dataset meaningfully, the current level of cleaning is okay for the analysis going forward.

4.3.6 Sentence Dataset Exploration

```
# Sentence count per transcript by filename
sentence_counts =
df.groupby("filename")["sentence_order"].max().reset_index(name="total_sent
ences")
print("\033[94m--- Sentence Count per Transcript ---\033[0m")
print(sentence_counts.describe())
print("\n")

# Compare average number of sentences between stances
stance_sentence_counts = df.groupby(["filename",
"stance"])["sentence_order"].max().reset_index()
avg_by_stance =
stance_sentence_counts.groupby("stance")["sentence_order"].mean()
print("\033[94m--- Average Sentence Count by Stance ---\033[0m")
print(avg_by_stance)
```

Code Snippet 24: Calculate Sentence Count Metrics

```
--- Sentence Count per Transcript ---
       total_sentences
count      3738.000000
mean         29.034778
std           4.736613
min          16.000000
25%          26.000000
50%          29.000000
75%          32.000000
max          56.000000


--- Average Sentence Count by Stance ---
stance
con    29.169525
pro    28.872717
Name: sentence_order, dtype: float64
```

Figure 32: Sentence Count Information

The average number of sentences per transcript is approximately 29, with most debates containing between 16 and 56 sentences. When comparing stances, con-side debaters average 29.17 sentences,

while pro-side debaters average 28.87. The difference is minimal, indicating no clear tendency or meaningful distinction in sentence count between the two sides.

```
# Plot distribution of number of sentences per transcript
plt.figure(figsize=(8, 4))
sns.histplot(data=sentence_counts, x="total_sentences", bins=20, kde=True,
color="skyblue")
plt.title("Distribution of Number of Sentences per Transcript")
plt.xlabel("Sentence Count")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```

Code Snippet 25: Create Sentence Count Histogram

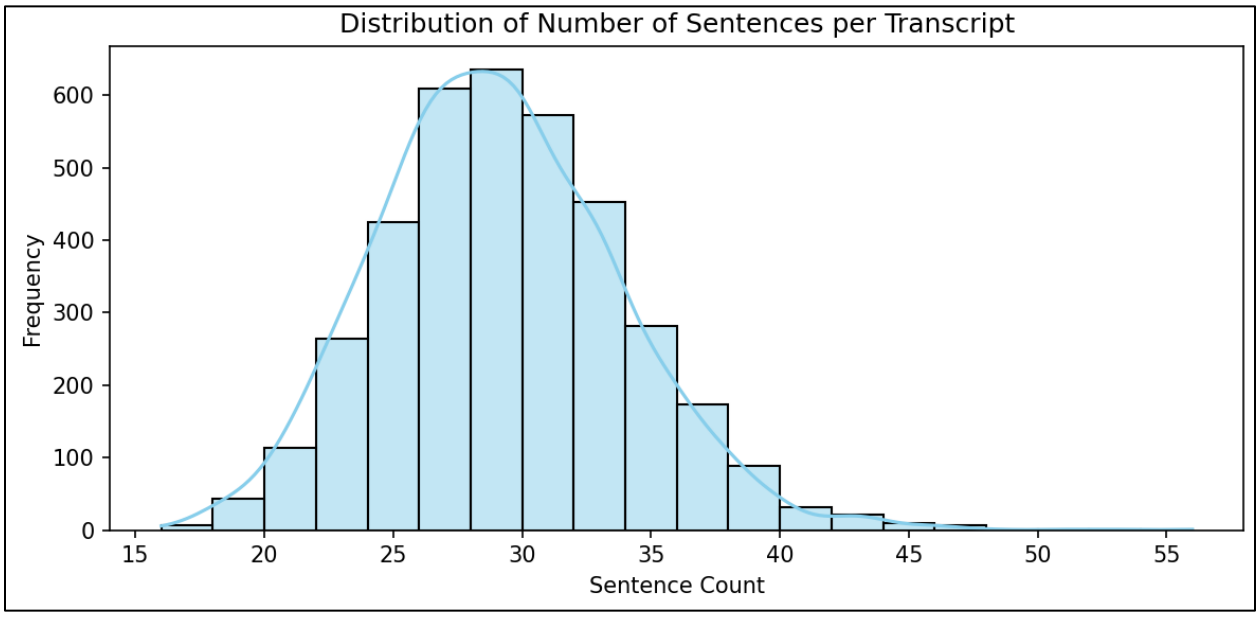


Figure 33: Sentence Count Distribution

The distribution of sentence counts per transcript is roughly standard, with most transcripts containing between 25 and 35 sentences. This suggests a consistent structure across debates. However, there is some variation, with a few transcripts containing as few as 16 sentences or as many as 56, indicating some variability in debate structure.

```
# Plot average sentence count by stance
plt.figure(figsize=(6, 4))
ax = sns.barplot(x=avg_by_stance.index, y=avg_by_stance.values,
palette="Set2")
plt.title("Average Sentence Count by Stance")
plt.ylabel("Average Sentence Count")
plt.xlabel("Stance")
for p in ax.patches:
    ax.annotate(f'{p.get_height():.1f}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='bottom', fontsize=10)
plt.tight_layout()
plt.show()
```

Code Snippet 26: Create Bar Chart of Average Stance

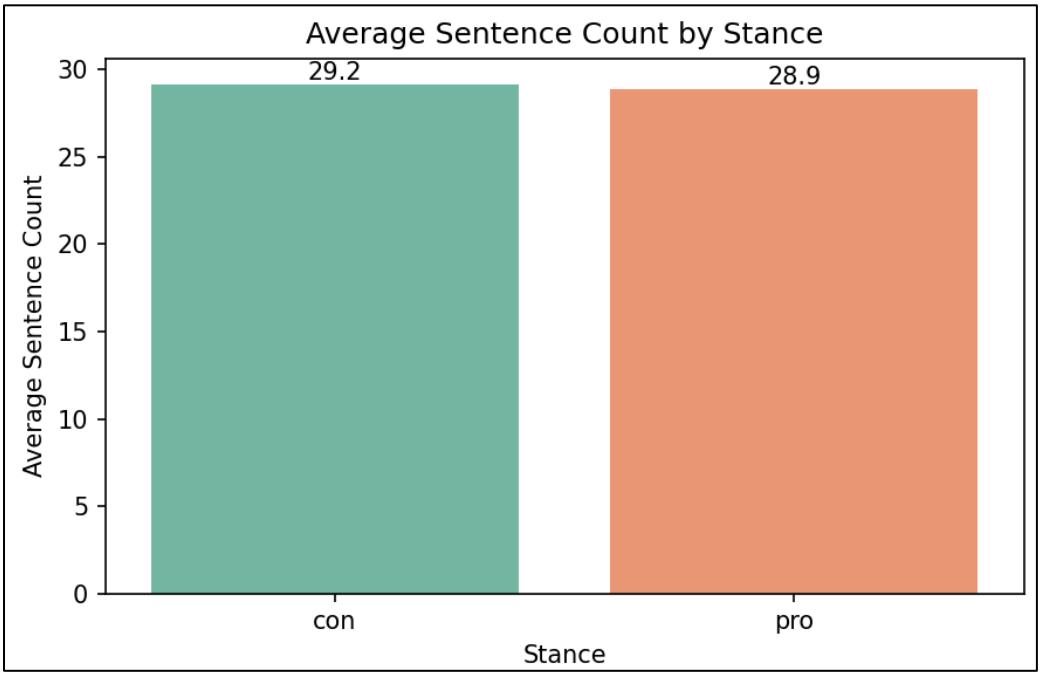


Figure 34: Bar Chart of Average Stance by Stance

The average sentence count between stances is nearly identical, with con-side debaters averaging 29.2 sentences and pro-side debaters averaging 28.9. This minimal difference suggests that stance has little to no impact on the overall quantity of speech in terms of sentence count.

```
# Save Dataset
df.to_csv("2_processed_data/processed_dataset.csv", index=False)
```

Code Snippet 27: Save Cleaned Dataset

The dataset is well-organised, and a cleaned version is then saved for further use and processing.

4.3.7 Further Text Exploration

This stage focuses on exploring the debate sentences in more depth rather than just visualising them through word clouds. Some steps include the common text pre-processing that is usually used in machine learning pipelines. However, they are not applied in this project for modelling because argument classification requires keeping the full wording intact, and small words can influence the structure of reasoning (Wilame, 2019). The dataset is further examined to look for meaningful patterns, frequent word sequences, and sentence structures. This helps to better understand the dataset and its linguistic features, enabling the identification of argument types, even if the model training handles tokenisation internally with its own tokeniser.

```
# Initialize stopwords and lemmatizer
stop_words = set(stopwords.words("english"))
lemmatizer = WordNetLemmatizer()

# POS tag mapping function (Penn Treebank → WordNet)
def get_wordnet_pos(treebank_tag):
    if treebank_tag.startswith('J'):
        return wn.ADJ
    elif treebank_tag.startswith('V'):
        return wn.VERB
    elif treebank_tag.startswith('N'):
        return wn.NOUN
    elif treebank_tag.startswith('R'):
        return wn.ADV
    else:
        return wn.NOUN

def preprocess_text(text):
    text = text.lower()
    tokens = word_tokenize(text)
    tagged_tokens = pos_tag(tokens)
    cleaned = []

    for word, tag in tagged_tokens:
        if word.isalpha() and word not in stop_words:
            wn_tag = get_wordnet_pos(tag)
            lemma = lemmatizer.lemmatize(word, wn_tag)
            cleaned.append(lemma)

    return cleaned

df["tokenized_sentence"] = df["sentence"].apply(preprocess_text)
```

Code Snippet 28: Remove Stop Words and Lemmatise Text

tokenized_sentence
['ban', 'boxing', 'boxing', 'blood', 'sport', 'absolutely', 'horrible', 'athlete', 'horrible', 'message', 'send', 'society']
['true']
['first', 'big', 'thing', 'box', 'horrific', 'consequence', 'people', 'participate']
['essentially', 'like', 'modern', 'day', 'equivalent', 'gladiator', 'battle', 'like', 'bad', 'way', 'possible']
['short', 'term', 'see', 'thing', 'like', 'concussion', 'see', 'thing', 'like', 'bone', 'break', 'people', 'face', 'someone', 'eye', 'get', 'like', 'gouge', 'damage', 'like', 'like', 'hit']
['kind', 'horrific', 'injury', 'people', 'obtain', 'short', 'term']
['lot', 'injury', 'specifically', 'also', 'long', 'term', 'impact']
['obviously', 'suffer', 'long', 'term', 'nerve', 'damage', 'suffer', 'brain', 'damage', 'chronic', 'hit']
['big', 'problem', 'sport', 'like', 'boxing', 'also', 'football', 'concussion', 'head', 'trauma', 'common']
['result', 'essentially', 'dementia', 'like', 'trait', 'horribly', 'young', 'age']

Figure 35: Text Output

The debate sentences are further processed to explore them for analysis. Each sentence is first converted to lowercase to maintain consistency and reduce duplicate tokens. NLTK's word_tokenize is used for tokenisation to split the text into individual words and symbols. After tokenisation, part-of-speech (POS) tagging is applied to identify each word's grammatical role. Then, stop words and non-alphabetic tokens are filtered out, as these are less useful when examining word-level patterns. Lastly, lemmatisation is performed with WordNet to convert words into their root forms based on their POS tags. The output of this process is stored in a new column called tokenized_sentence, which contains the cleaned tokens for each debate sentence. Although transformer-based models do not strictly require these steps, they help to understand the dataset, reveal common patterns, and support further sentence-level text exploration.

```
# N-gram Plot Function
def plot_ngrams(corpus, ngram_range=(1, 1), n=20, title="Top N-grams"):
    vec = CountVectorizer(ngram_range=ngram_range).fit(corpus)
    bag = vec.transform(corpus)
    sum_words = bag.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in
vec.vocabulary_.items()]
    top_words = sorted(words_freq, key=lambda x: x[1], reverse=True)[:n]

    if top_words:
        words, counts = zip(*top_words)
        plt.figure(figsize=(10, 5))
        ax = sns.barplot(x=list(counts), y=list(words), palette="crest")
        for i, v in enumerate(counts):
            ax.text(v + 1, i, str(v), color='black', va='center')
        plt.title(title)
        plt.xlabel("Frequency")
        plt.ylabel("N-gram")
        plt.tight_layout()
        plt.show()

# N-grams BEFORE stopword removal (raw sentence)
plot_ngrams(df["sentence"], (1, 1), 20, "Top 20 Unigrams (Raw)")
plot_ngrams(df["sentence"], (2, 2), 20, "Top 20 Bigrams (Raw)")
plot_ngrams(df["sentence"], (3, 3), 20, "Top 20 Trigrams (Raw)")

# Join tokens into strings
df["joined_tokens"] = df["tokenized_sentence"].apply(lambda x: " ".join(x))
df = df[df["joined_tokens"].str.strip() != ""]
# N-grams AFTER stopword removal (joined_tokens)
plot_ngrams(df["joined_tokens"], (1, 1), 20, "Top 20 Unigrams (Cleaned)")
plot_ngrams(df["joined_tokens"], (2, 2), 20, "Top 20 Bigrams (Cleaned)")
plot_ngrams(df["joined_tokens"], (3, 3), 20, "Top 20 Trigrams (Cleaned)")
```

Code Snippet 29: Create N-grams Visual

This part examines the most frequent word sequences (n-grams) within the debate sentences to uncover common language patterns. Bar graphs of the top 20 unigrams, bigrams, and trigrams are generated both before and after stop word removal. By comparing raw and cleaned results, it is possible to see which words, phrases, or combinations are most commonly used in debates and how these contribute to the overall argumentative style.

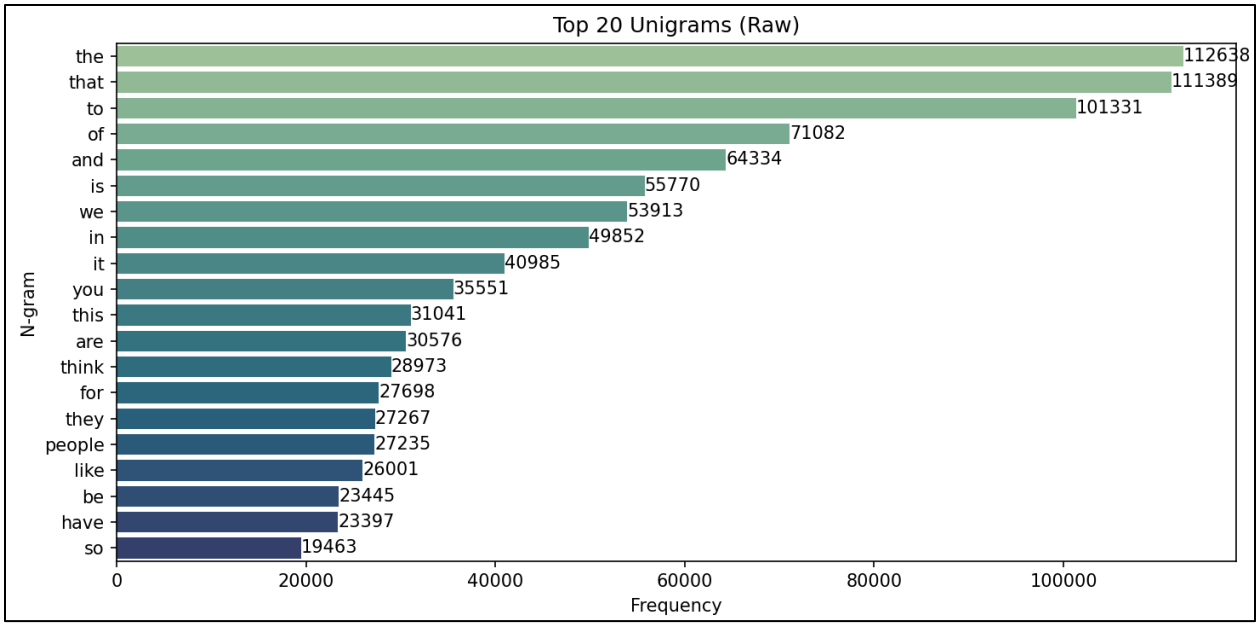


Figure 36: Top 20 Unigrams (Raw)

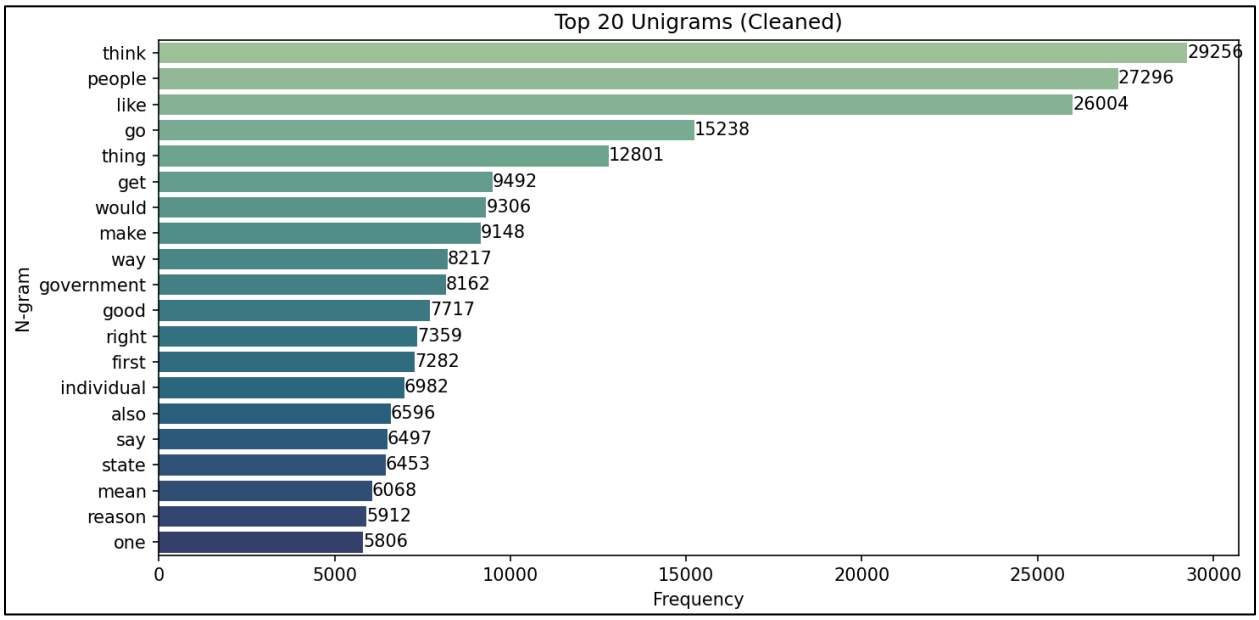


Figure 37: Top 20 Unigrams (Cleaned)

The most obvious difference is observed in the unigram word relevance before and after text cleaning. Before cleaning, the most common words are typical stop words such as "the", "that", and "to" that top the list based on their grammatical function, but are low on providing information about argument expression. Following the removal of stop words, the top unigrams represent more

content-bearing words, with terms such as "think", "people", "like", and "go" moving to the fore. These terms are more characteristic of debate terminology.

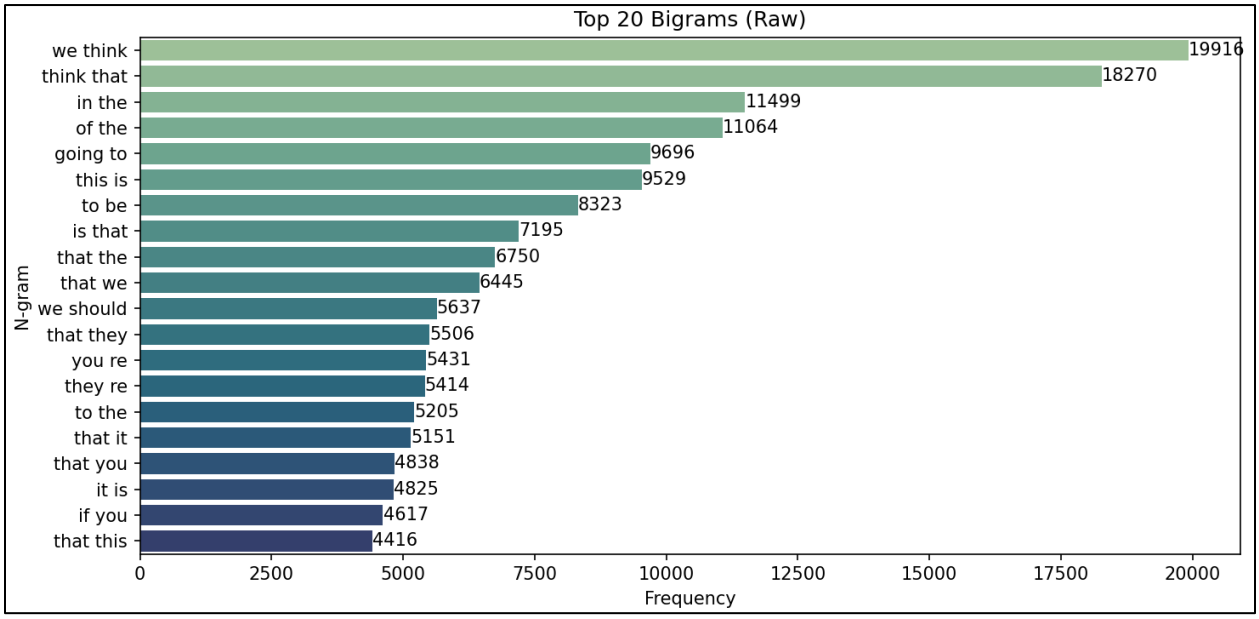


Figure 38: Top 20 Bigrams (Raw)

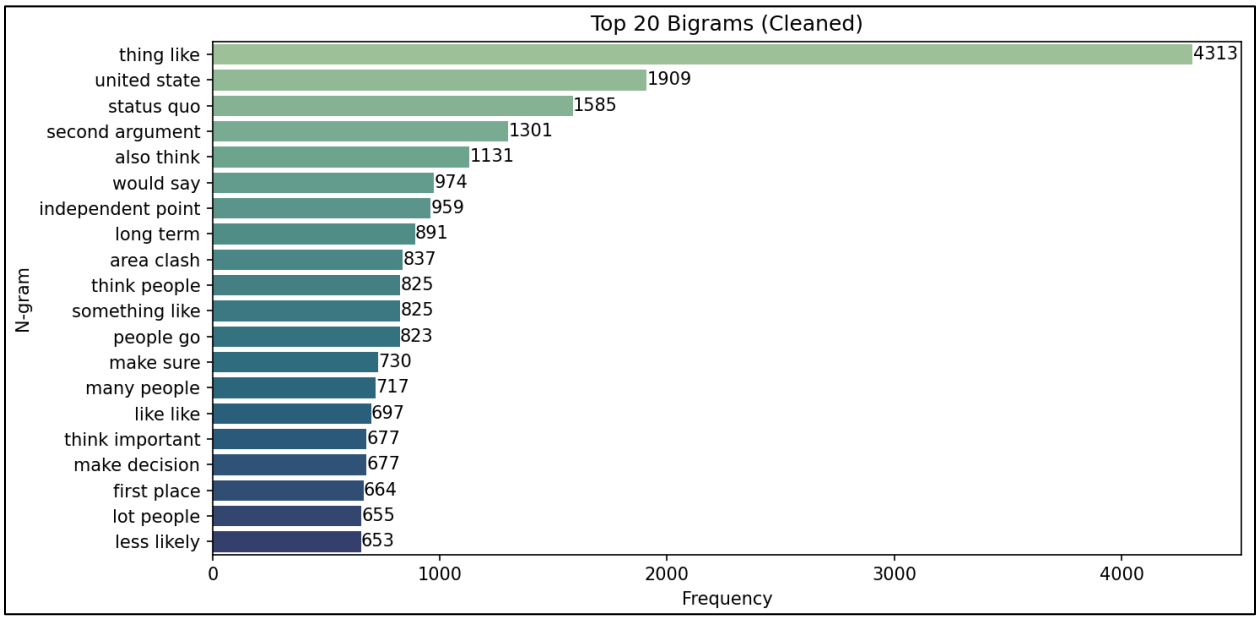


Figure 39: Top 20 Bigrams (Cleaned)

The bigram analysis provides a closer look at how ideas are expressed within debate sentences. In the raw text, frequent patterns such as “we think” and “think that” reflect the common style of

expressing opinions and framing arguments, while sequences like “in the” and “of the” represent structural connectors that shape how sentences are formed. In cleaned bigrams, “status quo,” “second argument,” and “united state” reveal themes related to policy, reasoning structure, and socio-political discussion, which are central in competitive debates. Other expressions like “make decision,” “long term,” and “independent point” show the argumentative nature of the discourse, which point out strategic reasoning and emphasis on evidence.

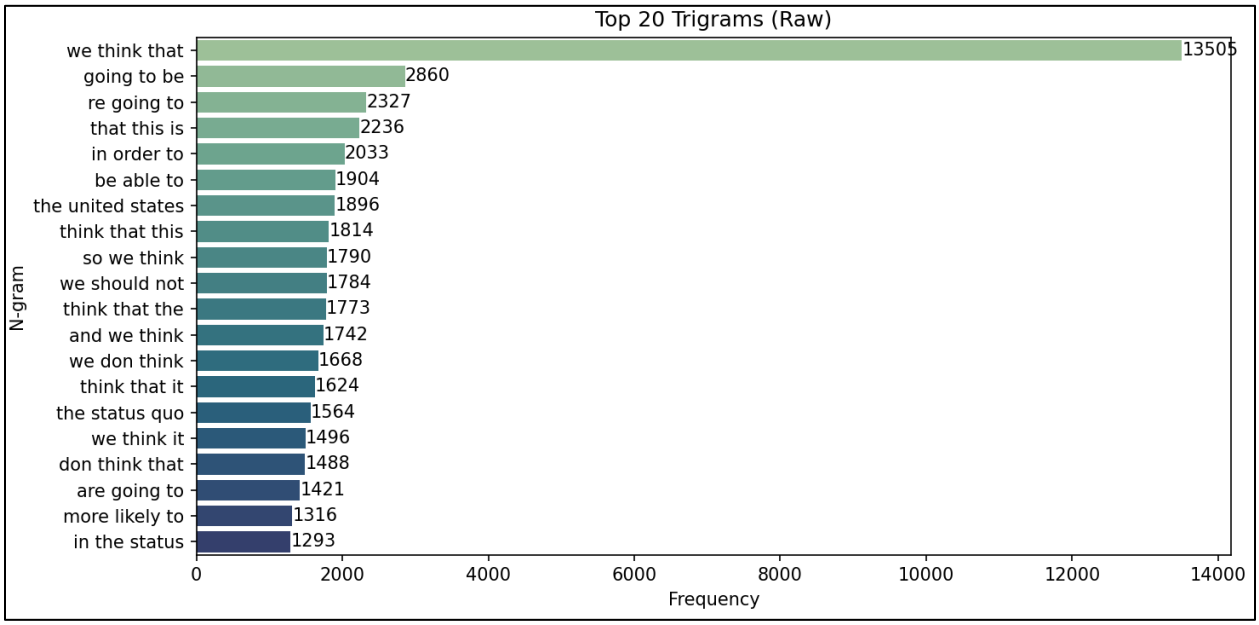


Figure 40: Top 20 Trigrams (Raw)

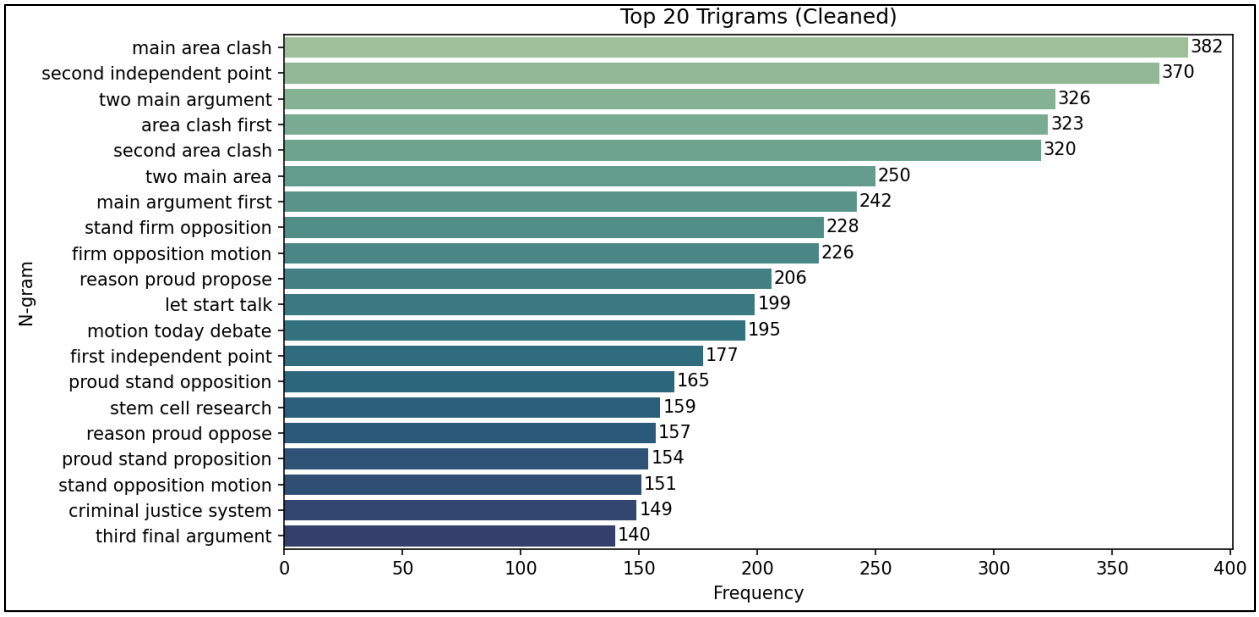


Figure 41: Top 20 Trigrams (Cleaned)

The trigram analysis shows how debaters construct more complex expressions and structured reasoning within their arguments. In the raw dataset, the most common trigrams, such as “we think that” and “going to be”, reflect general speech patterns and transitional phrases, which show how participants typically frame opinions or move between points. When examining more content-oriented trigrams, recurring expressions such as “main area clash,” “firm opposition motion,” and “second independent point” reveal the structured language of debate rounds, where speakers clearly mark the flow of arguments and counterarguments. Phrases like “criminal justice system” and “stem cell research” also point to specific debate topics, showing how certain domains are frequently discussed. Overall, trigram exploration uncovers both the routine scaffolding of spoken discourse and the structured, domain-specific terminology that shapes formal debating.

```
# Top Words by STANCE
def top_words_by_stance(df, stance_col, token_col, top_n=20):
    stance_groups = df.groupby(stance_col)[token_col]
    for stance, tokens in stance_groups:
        all_words = [word for sublist in tokens for word in sublist]
        top_words = Counter(all_words).most_common(top_n)

        if top_words:
            words, freqs = zip(*top_words)
            plt.figure(figsize=(10, 5))
            ax = sns.barplot(x=list(freqs), y=list(words), palette="Set2")
            for i, v in enumerate(freqs):
                ax.text(v + 1, i, str(v), color='black', va='center')
            plt.title(f"Top {top_n} Words for Stance: {stance}")
            plt.xlabel("Frequency")
            plt.ylabel("Words")
            plt.tight_layout()
            plt.show()

# If stance column exists
if "stance" in df.columns:
    top_words_by_stance(df, "stance", "tokenized_sentence")
```

Code Snippet 30: Create Top Words Visual

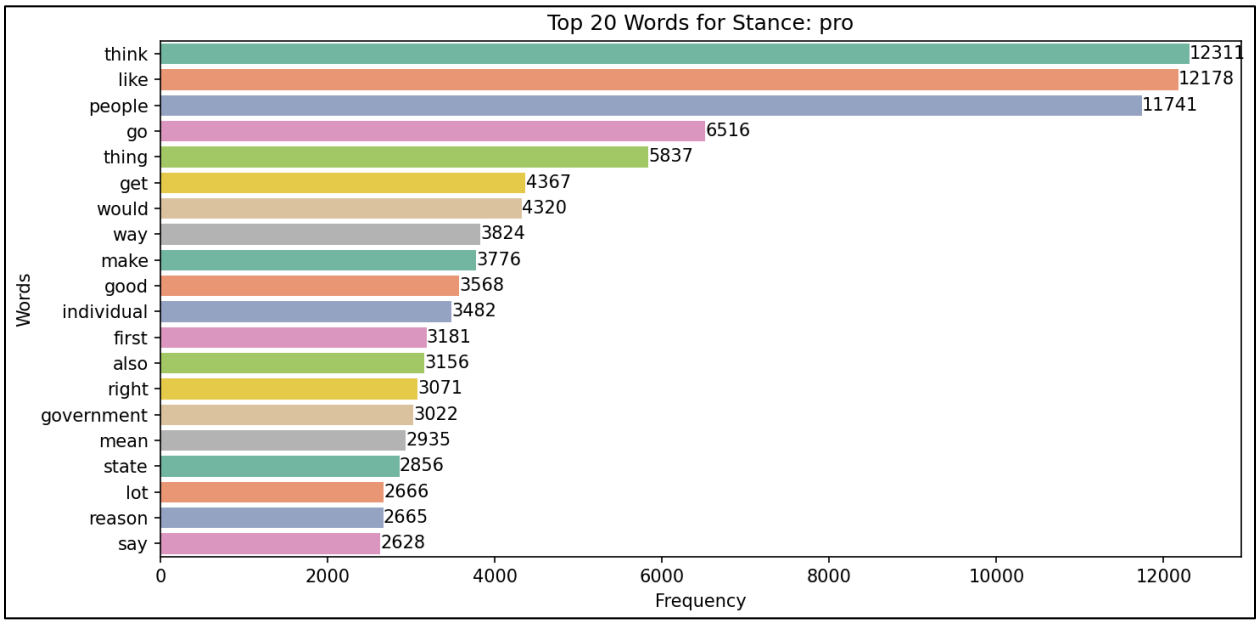


Figure 42: Top 20 Words by Pro

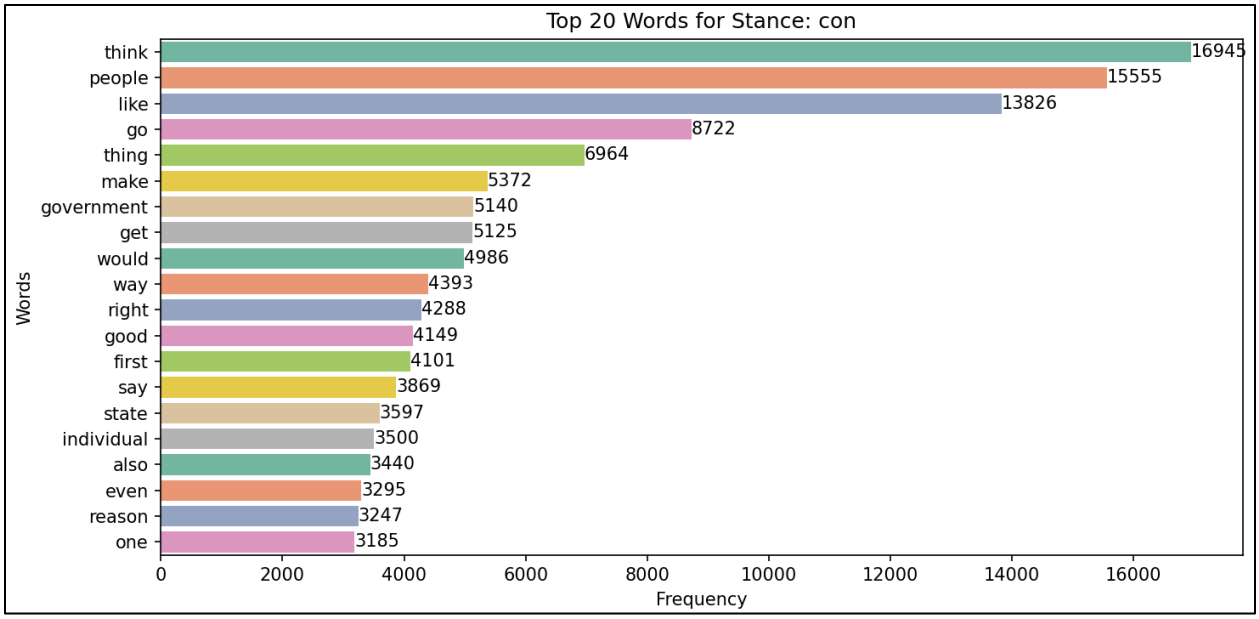


Figure 43: Top 20 Words by Con

The word frequency analysis by stance shows that both “pro” and “con” sides rely on general argumentative terms such as “think,” “people,” “like,” and “go.” These words appear across both stances, indicating that debaters share a common style of presenting opinions and structuring arguments, regardless of position. The absence of strongly stance-specific terms suggests that lexical usage remains relatively balanced and non-polarised between pro and con

.

4.3.8 Logical Fallacies Labelling Validation

```
# Initialize the model pipeline
model_path = "q3fer/distilbert-base-fallacy-classification"
pipe = pipeline("text-classification", model=model_path,
tokenizer=model_path)

# Load dataset
df = pd.read_csv("2_processed_data/3.1_text_preprocessed.csv")

# Create columns first with default empty values
df["fallacy"] = ""
df["fallacy_score"] = 0.0

# Loop through each sentence and assign predicted label and score
for idx, text in enumerate(df["sentence"]):
    result = pipe(text)
    df.at[idx, "fallacy"] = result[0]['label']
    df.at[idx, "fallacy_score"] = result[0]['score']
```

Code Snippet 31: Label Sentence with Fallacies

filename	stance	sentence	sentence_length	sentence_order	tokenized_sentence	fallacy	fallacy_score
DJ_121_ban-boxing_pro.trs.txt	pro	We should ban boxing because boxing is a blood sport and it's absolutely horrible for the athletes and it's a horrible message to send for society.	26	1	['ban', 'boxing', 'boxing', 'blood', 'sport', 'absolutely', 'horrible', 'athlete', 'horrible', 'message', 'send', 'society']	faulty generalization	0.646324635
DJ_121_ban-boxing_pro.trs.txt	pro	So why is this true?	5	2	['true']	circular reasoning	0.7511217594146729
DJ_121_ban-boxing_pro.trs.txt	pro	The first big thing is that boxing has horrific consequences on the people who participate in it.	17	3	['first', 'big', 'thing', 'box', 'horrific', 'consequence', 'people', 'participate']	faulty generalization	0.595710814
DJ_121_ban-boxing_pro.trs.txt	pro	It's essentially like a modern day equivalent of gladiator battles in like the worse way possible.	16	4	['essentially', 'like', 'modern', 'day', 'equivalent', 'gladiator', 'battle', 'like', 'bad', 'way', 'possible']	fallacy of logic	0.31098514795303345
DJ_121_ban-boxing_pro.trs.txt	pro	In the short term you see things like concussions, you see things like bones breaking in people's face, someone's eyes can get like gouged out and damage from like from like hits.	32	5	['short', 'term', 'see', 'thing', 'like', 'concussion', 'see', 'thing', 'like', 'bone', 'break', 'people', 'face', 'someone', 'eye', 'get', 'like', 'gouge', 'damage', 'like', 'like', 'hit']	faulty generalization	0.5822117924690247
DJ_121_ban-boxing_pro.trs.txt	pro	There's all kinds of horrific injuries that people can obtain in the short term.	14	6	['kind', 'horrific', 'injury', 'people', 'obtain', 'short', 'term']	false causality	0.382154644
DJ_121_ban-boxing_pro.trs.txt	pro	A lot of those injuries specifically can also of long term impacts.	12	7	['lot', 'injury', 'specifically', 'also', 'long', 'term', 'impact']	faulty generalization	0.6703944802284241

Figure 44: Output Dataset with Fallacies

The sentence-level dataset is enriched with logical fallacy annotations to evaluate whether the existing fallacy detection model can be used in the final debate analyser system. A pretrained Hugging Face model, q3fer/distilbert-base-fallacy-classification (Jin et al., 2022), is used through a text classification pipeline to automatically detect fallacies in each sentence. Two additional columns are created in the dataset: fallacy, which records the predicted fallacy type, and fallacy_score, which captures the model's confidence in that prediction. Each sentence is processed individually, and the model assigns the most likely fallacy label along with a probability

score. The result from this stage is a validation step to validate the model to be used for accurately annotating fallacies in the final deployed system.

```
# Apply confidence threshold
threshold = 0.7
df["fallacy"] = df.apply(lambda row: row["fallacy"] if
row["fallacy_score"] >= threshold else "no_fallacy", axis=1)

# Count fallacy types and calculate average score
fallacy_counts = df["fallacy"].value_counts().reset_index()
fallacy_counts.columns = ["fallacy", "count"]

fallacy_scores =
df.groupby("fallacy")["fallacy_score"].mean().reset_index()
fallacy_scores.columns = ["fallacy", "avg_score"]

# Merge count and score data
fallacy_stats = pd.merge(fallacy_counts, fallacy_scores, on="fallacy")
fallacy_stats = fallacy_stats.sort_values(by="count", ascending=False)

print(fallacy_stats)
```

Code Snippet 32: Measure Fallacy Label

	fallacy	count	avg_score
0	no_fallacy	75244	0.464899
1	faulty generalization	12553	0.869168
2	circular reasoning	6131	0.837571
3	fallacy of logic	4111	0.805256
4	appeal to emotion	2508	0.808763
5	fallacy of relevance	1906	0.783820
6	fallacy of extension	1431	0.836866
7	fallacy of credibility	1297	0.771160
8	false dilemma	1140	0.866039
9	ad populum	865	0.858530
10	false causality	862	0.828288
11	ad hominem	359	0.830905
12	intentional	125	0.788269

Figure 45: Fallacy Count and Average Score

Since the model assigns a fallacy label to every sentence based on the highest confidence score, a threshold of 0.7 is applied to filter out low-confidence predictions. Sentences with scores below this threshold are relabelled as "no_fallacy" to indicate insufficient evidence for a reliable fallacy classification. After filtering, most sentences (75,244) fall into this category. Among the confidently detected fallacies, the most common are faulty generalisation, circular reasoning, and the fallacy of logic, each showing strong average confidence scores above 0.80.

```
# Plot bar chart
plt.figure(figsize=(10, 6))
ax = sns.barplot(data=fallacy_stats, x="count", y="fallacy",
palette="Set3")
for container in ax.containers:
    ax.bar_label(container, fmt='%d', label_type='edge', fontsize=9)

plt.title("Detected Logical Fallacies (Filtered, Confidence ≥ 0.7)")
plt.xlabel("Sentence Count")
plt.ylabel("Fallacy Type")
plt.tight_layout()
plt.show()
```

Code Snippet 33: Create Bar Chart of Logical Fallacies

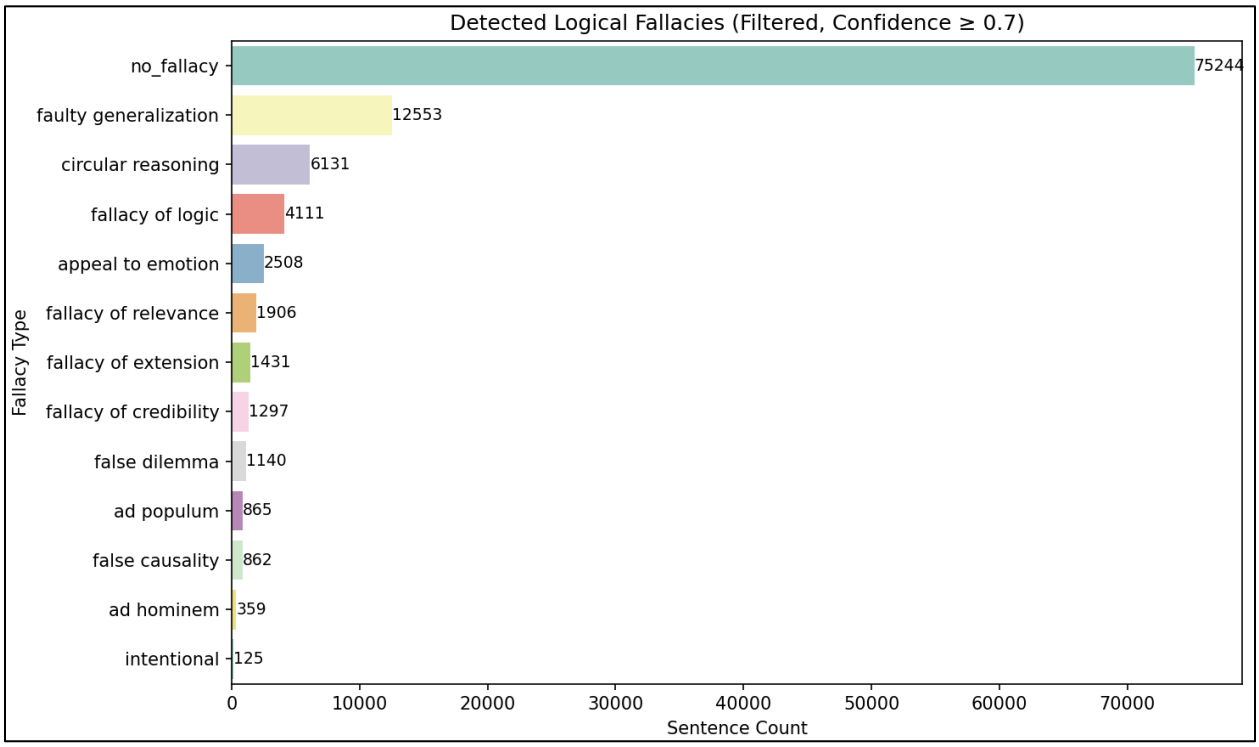


Figure 46: Logical Fallacies Bar Chart

The bar chart displays the distribution of detected logical fallacies after applying a confidence threshold of 0.7 to ensure label reliability. Most sentences over 75,000 are "no_fallacy," indicating a lack of strong evidence for fallacious reasoning. Among the confidently identified cases, faulty generalisation appears most frequently, followed by circular reasoning and fallacy of logic, occurring with notable frequency and consistently high confidence scores.

```
# Save filtered dataset
output_path =
"2_processed_data/processed_dataset_fallacies_validated.csv"
os.makedirs(os.path.dirname(output_path), exist_ok=True)
df.to_csv(output_path, index=False)
```

Code Snippet 34: Save Dataset

The dataset, with further text exploration and fallacy detection, labelling validation, and a score, is saved for logging and backup purposes.

4.4 Dataset Preparation

Before moving into model building and training, the debate dataset is not complete, which must first be prepared for argument type labels generation through transfer learning. Once labelling is done, text preprocessing is applied to convert sentences into formats suitable for different models. This phase ensures the dataset is complete and ready for modelling and evaluation.

4.4.1 Sampling

```
# Count sentence count per transcript
file_counts = df.groupby(["filename",
"stance"]).size().reset_index(name="sentence_count")

# Filter for transcripts with sentence count between 25 and 35
valid_files = file_counts[(file_counts["sentence_count"] >= 25) &
(file_counts["sentence_count"] <= 35)]

# Select first 75 pro and first 75 con
pro_files = valid_files[valid_files["stance"] == "pro"].head(75)["filename"]
con_files = valid_files[valid_files["stance"] == "con"].head(75)["filename"]

# Combine selected filenames
selected_files = pd.concat([pro_files, con_files])

# Filter original dataset to only selected files
selected_debates_df = df[df["filename"].isin(selected_files)]

# Keep only necessary columns
selected_debates_df = selected_debates_df[["filename", "sentence",
"sentence_order"]]

# Add empty columns for manual labeling
selected_debates_df["argument_type"] = ""
selected_debates_df["rationale"] = ""

df.head()

# Save final sampling dataset
output_path = "2_processed_data/sampling.csv"
selected_debates_df.to_csv(output_path, index=False)
```

Code Snippet 35: Generate Sample Dataset

filename	sentence	sentence_order	argument_type	rationale
DJ_121_ban-boxing_pro.trs.txt	We should ban boxing because boxing is a blood sport and it's absolutely horrible for the athletes and it's a horrible message to send for society.	1		
DJ_121_ban-boxing_pro.trs.txt	So why is this true?	2		
DJ_121_ban-boxing_pro.trs.txt	The first big thing is that boxing has horrific consequences on the people who participate in it.	3		
DJ_121_ban-boxing_pro.trs.txt	It's essentially like a modern day equivalent of gladiator battles in like the worse way possible.	4		
DJ_121_ban-boxing_pro.trs.txt	In the short term you see things like concussions, you see things like bones breaking in people's face, someone's eyes can get like gouged out and damage from like from like hits.	5		
DJ_121_ban-boxing_pro.trs.txt	There's all kinds of horrific injuries that people can obtain in the short term.	6		
DJ_121_ban-boxing_pro.trs.txt	A lot of those injuries specifically can also of long term impacts.	7		

Figure 47: Sampled Dataset

To prepare a subset of data for manual Toulmin argument labelling, the sentence-level dataset is filtered to include only transcripts with 25 to 35 sentences, ensuring moderate length and higher-quality content. From this pool, the first 75 transcripts with a "pro" stance and the first 75 with a "con" stance are selected, representing roughly 5% of the full dataset. The subset retains only essential columns (filename, sentence, and sentence_order), and two additional columns, argument_type and rationale, are initialised for manual annotation. The resulting sample is saved.

4.4.2 Labelling with ChatGPT and Verification

```
I would like you to manually label the argument components in a CSV file.

The dataset I uploaded contains sentences from debate transcripts, with each row
including:
- filename: the transcript file identifier
- sentence: one sentence from the transcript
- sentence_order: each sentence order from the transcript

Please label each sentence with only one of these argument roles:
- C: Claim (presenting an arguable idea you want the audience to accept)
- G: Grounds (be critical and ensure there should be evidence and reasoning that
support a claim or assertion in the sentence to label)
- W/Q: Warrant or Qualifier (warrant that shows a statement that connects a
reason to a claim, or qualifier that indicates the strength of the leap from the
data to the warrant and may limit how universally the claim applies)
- OTH: Others (Either the sentence is Backing, Rebuttal, or do not present any
argument type that has no meaning or like a question, or you are not clear or not
sure about the type to label)

Format your output by column:
- argument_type: the label (e.g., "C", "G", "W/Q", or "OTH")
Use GPT to generate so that you can see the previous and after sentence to be
more accurate. You need to provide short rationale to justify your label. Please
go through each sentence carefully because there can be different label even
there present similar keywords. After processing, output the dataset as a table.
```

Figure 48: Text for Prompting ChatGPT

filename	sentence	sentence_order	argument_type	rationale
DJ_121_ban-boxing_pro.trs.txt	We should ban boxing because boxing is a blood sport and it's absolutely horrible for the athletes and it's a horrible message to send for society.	1	C	This is the central proposition the speaker aims to prove — that boxing should be banned.
DJ_121_ban-boxing_pro.trs.txt	So why is this true?	2	OTH	This rhetorical question transitions from claim to reasoning, indicating a link between grounds and the claim.
DJ_121_ban-boxing_pro.trs.txt	The first big thing is that boxing has horrific consequences on the people who participate in it.	3	G	This provides factual support (evidence) for the claim — harmful consequences of boxing.
DJ_121_ban-boxing_pro.trs.txt	It's essentially like a modern day equivalent of gladiator battles in like the worse way possible.	4	W/Q	This analogy links the evidence (violence) to the argument that boxing is inherently harmful.
DJ_121_ban-boxing_pro.trs.txt	In the short term you see things like concussions, you see things like bones breaking in people's face, someone's eyes can get like gouged out and damage from like from like hits.	5	W/Q	Offers specific examples to reinforce the earlier ground about health consequences.
DJ_121_ban-boxing_pro.trs.txt	There's all kinds of horrific injuries that people can obtain in the short term.	6	OTH	Expands on the health consequences, providing further support for the grounds.
DJ_121_ban-boxing_pro.trs.txt	A lot of those injuries specifically can also of long term impacts.	7	W/Q	Continues detailing harm, reinforcing the evidence provided.

Figure 49: ChatGPT Output Labelling with Manual Verification

To streamline the manual labelling of argument components, a structured prompt is used for ChatGPT to assist in annotating sentences from debate transcripts. The prompt instructed ChatGPT to classify each sentence into one of four categories: Claim (C), Grounds (G), Warrant/Qualifier (W/Q), or Others (OTH). The OTH category includes elements such as Backing, Rebuttal, or sentences that do not clearly fit into the main Toulmin components. A short rationale for transparency in the classification process and validation accompanies each labelled sentence.

Unlike the full Toulmin model, which includes six elements (Claim, Grounds, Warrant, Backing, Qualifier, and Rebuttal), this project adopts a simplified scheme. The decision to group certain elements under OTH is to ensure consistency in labelling and reduce ambiguity between overlapping categories, such as Backing and Rebuttal. In practice, these elements are also less frequent, harder to distinguish reliably at the sentence level, and not always required for core argument structure identification. By focusing on the three primary roles (Claim, Grounds, Warrant/Qualifier) and consolidating the rest into OTH, the annotation process becomes more efficient. This makes the dataset clearer for training classification models.

The semi-automated approach with ChatGPT significantly reduced manual effort while maintaining human oversight, as the generated labels are reviewed and refined for accuracy. The validated labelled dataset serves as a foundation for training models through transfer learning, which are later applied to automate argument type detection across the full debate dataset.

4.4.3 Transfer Learning using DeBERTa

After the labelling of the sampling debate dataset was completed, the transfer learning stage focuses on building an argument classification model that can extend this annotation across the full dataset. It leverages knowledge from large-scale pretrained language models and adapts it to the specific task of identifying argument components. For this purpose, DeBERTa (Decoding-enhanced BERT with Disentangled Attention) is selected. DeBERTa is an advanced transformer model developed by Microsoft that improves upon BERT and RoBERTa by separating the position and content information in its attention mechanism, which enables the model to better capture the meaning and structure of sentences (*DeBERTa*, n.d.). By fine-tuning DeBERTa on the manually labelled sample, the model can then automatically label the remaining unlabelled debate sentences, which reduces the need for full manual annotation.

```
import pandas as pd
import numpy as np
import torch
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    Trainer,
    TrainingArguments,
    DataCollatorWithPadding
)
import optuna
from datasets import Dataset
from pathlib import Path
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,
accuracy_score, f1_score
```

Code Snippet 36: Import Libraries for Transfer Learning

```
# Load dataset
file_path = "2_processed_data/sampling_labelled.csv"
df = pd.read_csv(file_path, encoding="latin1")

valid = ['C', 'G', 'W/Q', 'OTH']
df = df[df['argument_type'].isin(valid)].copy()

df['input'] = df['sentence']
label_map = {label: idx for idx, label in enumerate(valid)}
inv_label_map = {idx: label for label, idx in label_map.items()}
df['label'] = df['argument_type'].map(label_map).astype(int)
```

Code Snippet 37: Load and Map Sampling Dataset

The transfer learning process begins with importing the required libraries and loading the labelled sample dataset. The text labels are converted into integers with a label map. For example, C→0, G→1, W/Q→2, OTH→3. This encoding is required because the DeBERTa classifier outputs numeric classes during training. The output above is a compact dataframe with one column for the raw sentence and one column with its numeric label.

```
# Stratified Split
train_texts, val_texts, train_labels, val_labels = train_test_split(
    df['input'].tolist(),
    df['label'].tolist(),
    test_size=0.2,
    stratify=df['label'],
    random_state=42
)
```

Code Snippet 38: Split Sampling Dataset

Next, the dataset is split into training and validation sets. A stratified split is used to ensure each argument type is represented fairly in both training and validation. The training set is for fine-tuning the DeBERTa model, while the validation set is held back to check performance on unseen data and prevent overfitting. In this project, 80% of the data is used for training and 20% for validation, with a fixed random seed to ensure consistency across runs.

```
# Tokenizer and Model
model_ckpt = "microsoft/deberta-v3-base"
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)
model = AutoModelForSequenceClassification.from_pretrained(
    model_ckpt,
    num_labels=len(valid)
)

# Tokenization
train_enc = tokenizer(train_texts, truncation=True, padding=True,
max_length=256)
val_enc = tokenizer(val_texts, truncation=True, padding=True,
max_length=256)
train_dataset = Dataset.from_dict({
    'input_ids': train_enc['input_ids'],
    'attention_mask': train_enc['attention_mask'],
    'labels': train_labels
})
val_dataset = Dataset.from_dict({
    'input_ids': val_enc['input_ids'],
    'attention_mask': val_enc['attention_mask'],
    'labels': val_labels
})
```

Code Snippet 39: Tokenise Input

Then, the tokeniser and model are set up for training. The DeBERTa-v3-base checkpoint from Microsoft is loaded with a pretrained tokeniser and model weights. The tokeniser breaks each debate sentence into tokens and applies truncation and padding to make sure all sequences fit a maximum length of 256 tokens. Both training and validation sentences are converted into token IDs and attention masks. These encoded inputs and numeric labels are wrapped into Hugging Face Dataset objects for mapping, which enable the Transformers pipeline to perform efficient batching and training.

```
# Metrics
def compute_metrics(pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)
    return {
        "accuracy": accuracy_score(labels, preds),
        "f1_macro": f1_score(labels, preds, average="macro")
    }

# Training Configuration
training_args = TrainingArguments(
    output_dir='logging/transfer_learning/results',
    overwrite_output_dir=True,
    do_train=True,
    do_eval=True,
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model='f1_macro',
    greater_is_better=True,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    learning_rate=2e-5,
    weight_decay=0.05,
    num_train_epochs=4,
    logging_dir='logging/transfer_learning/logs',
    logging_steps=10,
    save_total_limit=2,
    fp16=torch.cuda.is_available()
)
```

Code Snippet 40: Set Up Training Arguments

Before training the model, evaluation metrics and training configurations are defined. The metrics function calculates two key measures: accuracy and the macro-averaged F1 score. This ensures the model is not biased toward more frequent labels. The training arguments specify how the DeBERTa model should be fine-tuned. For the transfer learning model, training runs for four epochs with a batch size of 16, using a small learning rate of 0.00002 to make fine-tuned adjustments to the pre-trained weights. Weight decay helps to reduce overfitting, while evaluation and checkpoint saving occur at the end of each epoch. The best model is chosen based on the

highest macro F1 score, and mixed-precision training (fp16) is enabled to speed up computation. These settings are configured with consideration of balancing performance and computing burden.

```
# Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    tokenizer=tokenizer,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    data_collator=DataCollatorWithPadding(tokenizer),
    compute_metrics=compute_metrics
)

# Train
trainer.train()
```

Code Snippet 41: Train Transfer Learning Model

[864/864 27:19, Epoch 4/4]

Epoch	Training Loss	Validation Loss	Accuracy	F1 Macro
1	0.779200	0.689263	0.721578	0.578416
2	0.616600	0.632985	0.758701	0.710089
3	0.436200	0.623759	0.783063	0.719801
4	0.241000	0.687445	0.788863	0.729301

Figure 50: Transfer Learning Model Training Summary

The Trainer object combines the model, training configuration, datasets, tokeniser, data collator, and evaluation metrics into a pipeline. During training, both the training and validation performance are tracked at the end of each epoch. The results show improvements across four epochs. The training loss decreases consistently, while validation loss remains stable, indicating the model is learning without overfitting. Accuracy improves from 72.1% in the first epoch to 78.9% in the fourth, while the macro F1 score rises from 0.58 to 0.73.

```
# Final evaluation
preds = trainer.predict(val_dataset)
y_pred = np.argmax(preds.predictions, axis=1)

print("\n=== Classification Report ===")
labels_present = sorted(set(val_labels))
target_names_present = [inv_label_map[i] for i in labels_present]

print(classification_report(
    val_labels,
    y_pred,
    labels=labels_present,
    target_names=target_names_present,
    zero_division=0
))
```

Code Snippet 42: Print Transfer Learning Model Classification Report

```
=== Classification Report ===
              precision    recall  f1-score   support

           C       0.77      0.84      0.81       184
           G       0.83      0.79      0.81       255
         W/Q       0.80      0.86      0.83       339
         OTH       0.60      0.39      0.47        84

    accuracy                           0.79       862
   macro avg       0.75      0.72      0.73       862
weighted avg       0.78      0.79      0.78       862
```

Figure 51: Transfer Learning Model Classification Report

The transfer learning model evaluation is carried out using the validation dataset, and the results are summarised in the classification report. The overall accuracy reaches 79%, with a value of 78% in the last epoch of training, indicating no issues with overfitting or underfitting. The model performs strongly for Claim (C), Grounds (G), and Warrant/Qualifier (W/Q), with F1-scores around 0.81 to 0.83. However, performance is weaker for the OTH category, with an F1-score of 0.47, which shows the difficulty of learning patterns in the less consistent class. The macro average F1-score of 0.73 indicates that the model performs better in some categories than others, despite achieving high overall accuracy.

```
# True labels
y_true = val_labels

# Confusion matrix
cm = confusion_matrix(y_true, y_pred)
labels = ['C', 'G', 'W/Q', 'OTH']

# Print numeric matrix
print("Confusion Matrix:\n", cm)

# Display as heatmap
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=labels)
disp.plot(cmap=plt.cm.Blues, xticks_rotation=45)
plt.title("Confusion Matrix")
plt.tight_layout()
plt.show()
```

Code Snippet 43: Show Transfer Learning Model Confusion Matrix

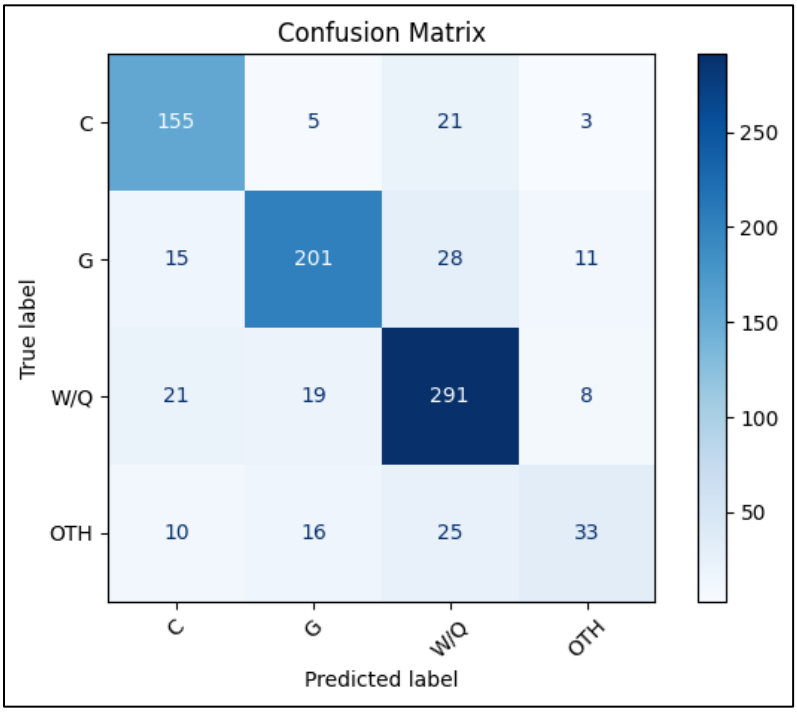


Figure 52: Transfer Learning Model Confusion Matrix

The confusion matrix shows the predictions with the true labels. Most Claim (C) sentences are correctly classified (155), with only small overlaps into other classes, similar to Grounds (G), with 201 correct predictions. However, some sentences are misclassified as W/Q, while Warrant or Qualifier has 291 correct predictions, and Others (OTH) has only 33 correct predictions. This confirms the classification report, which the Toulmin roles (C, G, W/Q) are learned well, while OTM remains less consistent.

```
def model_init():
    return AutoModelForSequenceClassification.from_pretrained(
        model_ckpt,
        num_labels=len(valid)
    )

def compute_metrics(pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)
    return {
        "accuracy": accuracy_score(labels, preds),
        "f1_macro": f1_score(labels, preds, average="macro")
    }

# Define search space for hyperparameter tuning
def model_hp_space(trial):
    return {
        "learning_rate": trial.suggest_float("learning_rate", 1e-6, 5e-5,
log=True),
        "num_train_epochs": trial.suggest_int("num_train_epochs", 2, 6),
        "per_device_train_batch_size":
trial.suggest_categorical("per_device_train_batch_size", [8, 16, 32]),
        "weight_decay": trial.suggest_float("weight_decay", 0.0, 0.3),
    }

# Trainer setup for hyperparameter search
trainer = Trainer(
    model_init=model_init,
    args=training_args,
    tokenizer=tokenizer,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    data_collator=DataCollatorWithPadding(tokenizer),
    compute_metrics=compute_metrics
)

# Launch Optuna search
best_trial = trainer.hyperparameter_search(
    direction="maximize",
    backend="optuna",
    hp_space=model_hp_space,
    n_trials=10,
    compute_objective=lambda metrics: metrics["eval_f1_macro"]
)

# Show best results
print("\n=== Best Hyperparameters ===")
print(best_trial)
```

Code Snippet 44: Implement Hyperparameter Tuning on Transfer Learning Model

```
=== Best Hyperparameters ===
BestRun(run_id='4', objective=0.7473599632727226,
hyperparameters={'learning_rate': 2.3663473935914246e-05,
'num_train_epochs': 4, 'per_device_train_batch_size': 8, 'weight_decay':
0.25923895271931247}, run_summary=None)
```

Figure 53: Best Hyperparameters for Transfer Learning Model

To further improve model performance, hyperparameter tuning is implemented using Optuna integrated with Hugging Face Trainer. The process systematically searches for the best values of key training parameters, including learning rate, number of epochs, batch size, and weight decay. Other training arguments remain the same as the first model training, and ten trials are run, with the best configuration achieving a macro F1 score of 0.747 using a learning rate of 0.0000237, 4 training epochs, 8 batch size, and 0.26 weight decay.

```
best_params = {
    'learning_rate': 2.3663473935914246e-05,
    'num_train_epochs': 4,
    'per_device_train_batch_size': 8,
    'weight_decay': 0.25923895271931247
}

best_training_args = TrainingArguments(
    output_dir='logging/transfer_learning/best_run',
    overwrite_output_dir=True,
    do_train=True,
    do_eval=True,
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model='f1_macro',
    greater_is_better=True,
    learning_rate=best_params['learning_rate'],
    num_train_epochs=best_params['num_train_epochs'],

per_device_train_batch_size=best_params['per_device_train_batch_size'],
    per_device_eval_batch_size=16,  # Can remain fixed
    weight_decay=best_params['weight_decay'],
    logging_dir='logging/transfer_learning/logs_best',
    logging_steps=10,
    save_total_limit=2,
    fp16=torch.cuda.is_available()
)

# Reinitialize model with same architecture
model = AutoModelForSequenceClassification.from_pretrained(
    model_ckpt,
    num_labels=len(valid)
)

# Define Trainer with best hyperparameters
best_trainer = Trainer(
    model=model,
    args=best_training_args,
    tokenizer=tokenizer,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    data_collator=DataCollatorWithPadding(tokenizer),
    compute_metrics=compute_metrics
)

best_trainer.train()
```

Code Snippet 45: Retrain Transfer Learning Model with Best Hyperparameters

[1724/1724 15:05, Epoch 4/4]

Epoch	Training Loss	Validation Loss	Accuracy	F1 Macro
1	0.546600	0.656864	0.730858	0.610144
2	0.521100	0.523691	0.806265	0.752300
3	0.307900	0.551608	0.820186	0.766877
4	0.089700	0.746809	0.821346	0.759881

Figure 54: Transfer Learning Model Training (Best Tuned)

After that, the best hyperparameters found are used to retrain the transfer learning model. Similary, evaluation and checkpointing run at every epoch, with the best model selected by macro F1. There are improvements over the baseline, with accuracy rising to about 82%, and macro F1 to 0.76. Validation loss dips by epoch 2 to 3 before pushing up, but *load_best_model_at_end=True* keeps the checkpoint with the highest F1. This tuned model is the version used to auto-label the remaining unlabelled sentences.

```
=== Classification Report ===
              precision    recall  f1-score   support

           C       0.88      0.85      0.87       184
           G       0.83      0.83      0.83       255
         W/Q       0.85      0.88      0.86       339
         OTH       0.51      0.50      0.51        84

    accuracy                           0.82       862
   macro avg       0.77      0.76      0.77       862
weighted avg       0.82      0.82      0.82       862
```

Figure 55: Transfer Learning Model Classification Report (Best Tuning)

The tuned model shows the overall accuracy improved to 82% with a macro F1 score of 0.77. The OTH category remains the weakest, with an F1 score of 0.51, which reflects the inherent difficulty of grouping diverse cases, but has improved since the first training. Compared with the baseline model, the tuned version has higher precision and recall for the core categories.

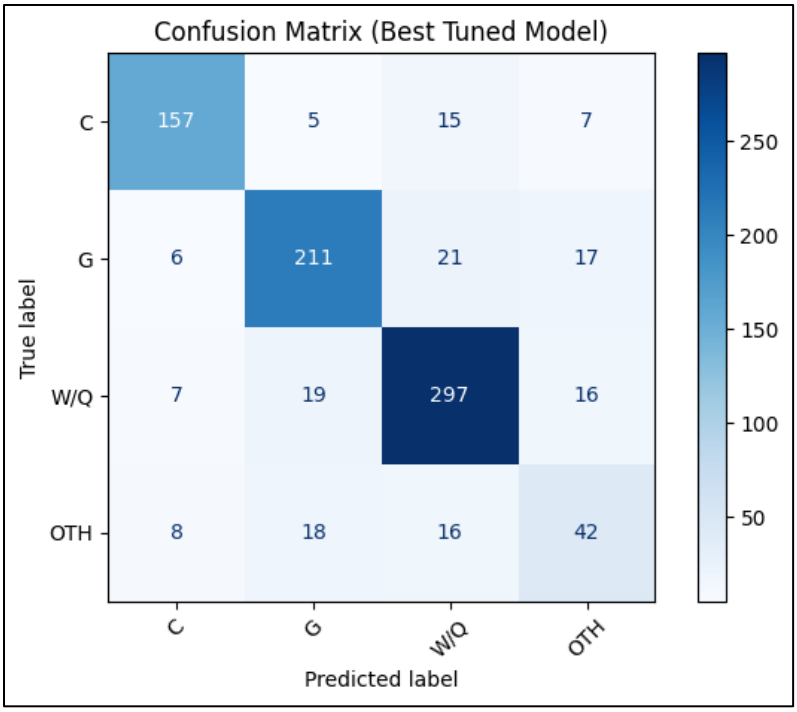


Figure 56: Transfer Learning Model Confusion Matrix (Best Tuning)

The confusion matrix shows Claim (C) has 157 correct classifications, 211 for Grounds (G), 297 for Warrant or Qualifies (W/Q), and the Others (OTH) category remains the most difficult, with 42 correct predictions out of 84 and frequent misclassification into G and W/Q. In short, the tuned transfer learning DeBERTa model improves on the baseline across all classes, especially Grounds and Others.

```
# Save final model and tokenizer
save_path = "3_model/transfer-learning/deberta-v3-tuned-final"
model.save_pretrained(save_path)
tokenizer.save_pretrained(save_path)
```

Code Snippet 46: Save Transfer Learning Model

Lastly, the best-tuned DeBERTa transfer learning model and its tokeniser are saved into a directory so that they can be reloaded later without retraining. It becomes ready for downstream tasks such as automatically labelling the full debate dataset at the next stage.

4.4.4 Labelling Whole Dataset using Transfer Learning Model

In this stage, the previously saved best-tuned transfer learning DeBERTa model is applied to the full debate dataset. The model will automatically predict and assign argument type labels (C, G, W/Q, OTH) to every remaining sentence. While this model achieves an accuracy of approximately 82%, it does not fully correctly label all argument types. To address this, the labelled dataset is reviewed and revised if needed to improve reliability. However, due to the large size of the dataset, it is not feasible to manually check every label perfectly, which represents a practical constraint of this project.

```
import pandas as pd
import torch
from transformers import AutoTokenizer,
AutoModelForSequenceClassification
from datasets import Dataset
from torch.nn.functional import softmax
import numpy as np
from tqdm import tqdm
```

Code Snippet 47: Import Libraries for Labelling

```
# Load saved model
model_path = "3_model/transfer-learning/deberta-v3-tuned-final"
tokenizer = AutoTokenizer.from_pretrained(model_path)
model = AutoModelForSequenceClassification.from_pretrained(model_path)
model.eval()

if torch.cuda.is_available():
    model.cuda()

# Load datasets
df_labeled = pd.read_csv("2_processed_data/sampling_labelled.csv",
encoding='latin1')
df_full = pd.read_csv("2_processed_data/processed_dataset.csv",
encoding='latin1')
```

Code Snippet 48: Load Transfer Learning Model and Dataset

The process begins by loading the previously saved transfer learning model and tokeniser. The model is set to evaluation mode to ensure it is used for prediction only without further training. This model will utilise CUDA to accelerate inference across the large dataset. There are two datasets loaded, including a manually labelled sample and the unlabelled processed dataset.

```
# Map known labels to full dataset
valid_labels = ['C', 'G', 'W/Q', 'OTH']
label_map = {label: i for i, label in enumerate(valid_labels)}
inv_label_map = {v: k for k, v in label_map.items()}

# Ensure clean types
df_labeled = df_labeled[df_labeled['argument_type'].isin(valid_labels)].copy()
df_labeled['argument_type'] = df_labeled['argument_type'].astype(str)
df_full['sentence'] = df_full['sentence'].astype(str)

# Assign known labels from sampling to processed dataset
df_full['argument_type'] =
df_full['sentence'].map(df_labeled.drop_duplicates(subset='sentence').set_index(
'sentence')['argument_type'])
```

Code Snippet 49: Map and Paste Known Label First

Then, the known argument types from the annotated sample are first aligned with the processed dataset. A label map is created for valid classes to have consistent indexing and decoding of predictions later. Then, the sentences in the full dataset are cross-checked against the sampling dataset, and where a match is found, the known human-assigned label is directly copied over. This ensures that any sentence already labelled in the sample keeps its verified argument type.

```
# Identify sentences to predict
df_to_predict = df_full[df_full['argument_type'].isna()].copy()
sentences_to_predict = df_to_predict['sentence'].tolist()

# Predict using the model
batch_size = 16
predicted_labels = []

for i in tqdm(range(0, len(sentences_to_predict), batch_size)):
    batch_sentences = sentences_to_predict[i:i+batch_size]
    encodings = tokenizer(batch_sentences, padding=True, truncation=True,
return_tensors="pt", max_length=256)
    if torch.cuda.is_available():
        encodings = {k: v.cuda() for k, v in encodings.items()}

    with torch.no_grad():
        outputs = model(**encodings)
        probs = softmax(outputs.logits, dim=1)
        preds = torch.argmax(probs, dim=1).cpu().numpy()
        predicted_labels.extend(preds)

# Update predictions back to dataframe
df_full.loc[df_full['argument_type'].isna(), 'argument_type'] =
[inv_label_map[p] for p in predicted_labels]

# Save the final fully labeled dataset
df_full.to_csv("2_processed_data/labelled_dataset.csv", index=False)
```

Code Snippet 50: Label Remaining Sentences

The model starts by labelling the remaining unannotated sentences. The argument type labelling is performed in batches of 16 sentences at a time to efficiently handle the large dataset. The model takes sentence input and outputs all four argument types with probabilities, and the label with the highest probability is selected. The labels are mapped back to the argument type codes (C, G, W/Q, OTH). The labelled dataset is saved.

filename	stance	sentence	sentence_length	sentence_order	argument_type
RG_4005_chemical-weapon_con_JL_implicit.trs.txt	con	Chemical weapons should be banned.	5	1	C
RG_4005_chemical-weapon_con_JL_implicit.trs.txt	con	On this whole argument that james has going about how he's going to magically regulate the use of chemical weapons, I want him to actually have to provide evidence on how this is going to happen.	36	2	W/Q
RG_4005_chemical-weapon_con_JL_implicit.trs.txt	con	Like same as with weapons of mass destruction, once you allow the existence, regulation becomes a country by country agreement with no real way of knowing anything for sure.	29	3	W/Q
RG_4005_chemical-weapon_con_JL_implicit.trs.txt	con	Like we see this in the status quo when the united states keeps trying to negotiate iran being non-nuclear.	19	4	G
RG_4005_chemical-weapon_con_JL_implicit.trs.txt	con	What happens every time is that every time we get close to an agreement we find video footage that iran is actually lying to us or iran pulls back or something similar.	32	5	W/Q

Figure 57: Completed Labelled Dataset by Transfer Learning Model

The figure above shows the output of the transfer learning model after being applied to the processed dataset. Each sentence is automatically assigned an argument type label (C, G, W/Q, OTH).

filename	stance	sentence	sentenc	sentence_ord	predicted_argument_ty	revised_argument_ty
DJ_841_blasphemy_pro.trs.txt	pro	So, let's start by talking about what exactly we're going to criminalize, because we're not going to criminalize every single kind of blasphemy.	23	2	G	OTH
DJ_841_blasphemy_pro.trs.txt	pro	Let's start by talking about why these things are necessary.	10	6	W/Q	OTH
DJ_841_blasphemy_pro.trs.txt	pro	So, let's talk about why blasphemy needs this kind of a qualification.	12	12	W/Q	OTH
DJ_841_blasphemy_pro.trs.txt	pro	But, the next thing that I want to talk about is about the public disorder that can happen, for these kinds of things.	23	17	OTH	W/Q
DJ_881_holocaust-denial_pro.trs.txt	pro	So let's start by talking about why I think that holocaust denial just should be a crime.	17	2	C	OTH

Figure 58: Completed Labelled Dataset (Revised)

While the model provides consistent predictions across the dataset, it is worth noting that its accuracy is approximately 82%. To improve the reliability, a manual rough quick revision process is conducted to check and correct the incorrectly predicted labels. In total, 7,060 labels out of 108,532 sentences were revised to improve quality, while acknowledging the constraint of not being able to review the entire dataset carefully due to its large size. This approach balances scalability with accuracy. The processed dataset is now labelled and ready for modelling.

4.4.5 Text Pre-processing

With the completed labelled dataset, each sentence with its argument type label is converted into the required input structure, to be ready for training and evaluation.

4.4.5.1 SVM

```
texts = df[TEXT_COL].astype(str).tolist()
labels_raw = df[LABEL_COL].astype(str).tolist()

# Encode labels
le = LabelEncoder()
y = le.fit_transform(labels_raw)
id2label = {i: lbl for i, lbl in enumerate(le.classes_)}
label2id = {lbl: i for i, lbl in id2label.items()}

# Split
X_train, X_val, y_train, y_val = train_test_split(
    texts, y, test_size=0.2, stratify=y, random_state=42
)
```

Code Snippet 51: Process Dataset for SVM

Since SVM requires numeric labels, a label encoder is applied to convert the argument type classes into integer values, with mapping recorded for later decoding. Next, the dataset is split into training and validation sets using an 80/20 ratio, and the distribution of labels is preserved through stratification.

```
# Build pipeline (TF-IDF + LinearSVC (calibrated))
base_svc = LinearSVC(C=1.0, class_weight="balanced", random_state=42)
clf = CalibratedClassifierCV(base_estimator=base_svc, method="sigmoid", cv=3)

pipeline = Pipeline([
    ("tfidf", TfidfVectorizer(
        lowercase=True,
        strip_accents="unicode",
        analyzer="word",
        ngram_range=(1, 2),
        max_features=30000,
        min_df=2
    )),
    ("clf", clf),
])

# Train
pipeline.fit(X_train, y_train)
```

Code Snippet 52: Use TF-IDF to Process Text

The raw sentences are then transformed into numeric features using a TF-IDF vectorizer. This process will lowercase text, strip accents, and build a sparse vocabulary of unigrams and bigrams (n=1, 2) to keep terms that appear at least twice (min_df=2), and cap the feature space at 30,000 terms. TF-IDF downweighs common tokens and emphasises words and phrases that are distinctive to each argument type (GeeksforGeeks, 2025g). It is implemented inside a pipeline so that feature extraction and model training run together consistently on both training and validation sets.

4.4.5.2 CNN

```
label2id = {l:i for i,l in enumerate(sorted(df[LABEL_COL].unique()))}
id2label = {i:l for l,i in label2id.items()}

y_all = df[LABEL_COL].map(label2id).astype(int).tolist()
X_all = df[TEXT_COL].astype(str).tolist()

X_train, X_val, y_train, y_val = train_test_split(
    X_all, y_all, test_size=0.2, stratify=y_all, random_state=SEED
)

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased", use_fast=True)
if tokenizer.pad_token_id is None:
    tokenizer.pad_token = tokenizer.eos_token
```

Code Snippet 53: Process Dataset for CNN

To process text for CNN, the argument type labels are mapped into integer IDs, stored as integer lists, while sentences are stored as text inputs. Then, it is split into training and validation sets with an 80/20 stratified split. A pretrained BERT tokeniser is used to tokenise the sentences into subword units because CNN needs fixed-length numerical input. The padding tokens are also defined to align to the same maximum length.

4.4.5.3 DeBERTa

```
label_map = {label: idx for idx, label in enumerate(valid)}
inv_label_map = {idx: label for label, idx in label_map.items()}
df['label'] = df['revised_argument_type'].map(label_map).astype(int)

# Stratified Split
train_texts, val_texts, train_labels, val_labels = train_test_split(
    df['input'].tolist(),
    df['label'].tolist(),
    test_size=0.2,
    stratify=df['label'],
    random_state=42
)

# Tokenizer and Model
model_ckpt = "microsoft/deberta-v3-base"
tokenizer = AutoTokenizer.from_pretrained(model_ckpt, use_fast=True)


# Tokenization
train_enc = tokenizer(train_texts, truncation=True, padding=True, max_length=256)
val_enc = tokenizer(val_texts, truncation=True, padding=True, max_length=256)
train_dataset = Dataset.from_dict({
    'input_ids': train_enc['input_ids'],
    'attention_mask': train_enc['attention_mask'],
    'labels': train_labels
})
val_dataset = Dataset.from_dict({
    'input_ids': val_enc['input_ids'],
    'attention_mask': val_enc['attention_mask'],
    'labels': val_labels
})
```

Code Snippet 54: Process Dataset for DeBERTa

For the DeBERTa-v3 model, the labels are also mapped into integers first because the model's classifier head outputs numerical class predictions. The dataset is then split into training and validation sets using a stratified approach, like SVM and CNN, to ensure consistency. A pretrained DeBERTa-v3-base tokeniser is applied to convert the sentences into token IDs and attention masks. Padding and truncation are used with a maximum of 256 tokens. The resulting encoded data is packaged into Hugging Face Dataset objects.

4.4.5.4 GPT

```
label_map = {label: i for i, label in enumerate(valid_labels)}
inv_label_map = {i: label for label, i in label_map.items()}
df['label'] = df['revised_argument_type'].map(label_map).astype(int)

# Stratified Split
train_texts, val_texts, train_labels, val_labels = train_test_split(
    df['sentence'].tolist(),
    df['label'].tolist(),
    test_size=0.2,
    stratify=df['label'],
    random_state=42
)

# Tokenizer and Model
model_ckpt = "gpt2"

tokenizer = GPT2Tokenizer.from_pretrained(model_ckpt)
tokenizer.pad_token = tokenizer.eos_token

# Define tokenization function
def tokenize(batch):
    return tokenizer(
        batch["text"],
        padding="max_length",
        truncation=True,
        max_length=256,
        return_tensors=None
    )

# Create Hugging Face Dataset objects
train_ds = Dataset.from_dict({
    "text": train_texts,
    "label": train_labels
}).map(tokenize, batched=True)

val_ds = Dataset.from_dict({
    "text": val_texts,
    "label": val_labels
}).map(tokenize, batched=True)
```

Code Snippet 55: Process Dataset for GPT

The text processing preparation for the GPT-2 model training is similar to the approach used in DeBERTa-v3. The main difference lies in the use of the GPT-2 tokeniser, which is specifically designed for autoregressive language models. The end-of-sequence token is assigned to handle padding, as GPT-2 does not include a default padding token. The sentences are also tokenised with truncation and padding to a maximum length of 256 tokens.

4.4.5.5 DeepSeek

```
MODEL_CKPT = "deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B"

label2id = {label: i for i, label in enumerate(VALID_LABELS)}
id2label = {i: label for label, i in label2id.items()}
df["label"] = df[LABEL_COL].map(label2id).astype(int)

train_df, val_df = train_test_split(
    df[[TEXT_COL, "label"]].copy(),
    train_size=0.8,
    stratify=df["label"],
    random_state=42,
)

# Tokenizer
tokenizer = AutoTokenizer.from_pretrained(MODEL_CKPT, use_fast=True)
if tokenizer.pad_token_id is None and tokenizer.eos_token_id is not None:
    tokenizer.pad_token_id = tokenizer.eos_token_id
tokenizer.padding_side = "right"

def tokenize_batch(batch):
    return tokenizer(
        batch[TEXT_COL],
        truncation=True,
        max_length=256,
    )

train_ds = Dataset.from_pandas(train_df.reset_index(drop=True)).map(
    tokenize_batch, batched=True, remove_columns=[TEXT_COL]
)
val_ds = Dataset.from_pandas(val_df.reset_index(drop=True)).map(
    tokenize_batch, batched=True, remove_columns=[TEXT_COL]
)
```

Code Snippet 56: Process Dataset for DeepSeek

The text preprocessing for DeepSeek-R1 fine-tuning follows the same workflow as that used for DeBERTa-v3 and GPT-2, including dataset splitting and label mapping. The difference is using the DeepSeek/Qwen-compatible tokeniser from the chosen checkpoint. Inputs are tokenised with truncation and padding to a maximum length of 256 tokens as well.

4.4.5.6 LLaMA

```
label2id = {l:i for i,l in enumerate(VALID)}
id2label = {i:l for l,i in label2id.items()}
df["label"] = df[LABEL_COL].map(label2id).astype(int)

train_texts, val_texts, train_labels, val_labels = train_test_split(
    df[TEXT_COL].astype(str).tolist(),
    df["label"].tolist(),
    test_size=0.2, stratify=df["label"], random_state=42
)

# Model and Tokenizer
model_ckpt = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"

tokenizer = AutoTokenizer.from_pretrained(model_ckpt, use_fast=True,
trust_remote_code=True)
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = "right"

MAX_LEN = 96

def tokenize(batch):
    return tokenizer(
        batch["text"],
        truncation=True,
        padding="max_length",
        max_length=MAX_LEN
    )

train_ds = Dataset.from_dict({"text": train_texts, "label": train_labels}).map(
    tokenize, batched=True, num_proc=1, remove_columns=["text"]
)
val_ds = Dataset.from_dict({"text": val_texts, "label": val_labels}).map(
    tokenize, batched=True, num_proc=1, remove_columns=["text"]
)

cols = ["input_ids", "attention_mask", "label"]
train_ds = train_ds.with_format(type="torch", columns=cols)
val_ds   = val_ds.with_format(type="torch", columns=cols)
```

Code Snippet 57: Process Dataset for LLaMA

TinyLLaMA also possess similar processing to other transformer setups, but uses a smaller checkpoint and a shorter max length due to limited compute. TinyLLaMA is chosen instead of full LLaMA, and the sequence is capped at 96 to ensure the training remains feasible for a six GB GPU without out-of-memory (OOM) errors.

4.5 Model Building

This phase develops and trains the different models to classify argument types within debate sentences by using the prepared dataset. Machine learning, deep learning, and transformer architectures are built and fine-tuned to identify argument types, including Claims, Grounds, Warrants /Qualifiers, and Others.

4.5.1 SVM

```
# Build pipeline (TF-IDF + LinearSVC (calibrated))
base_svc = LinearSVC(C=1.0, class_weight="balanced", random_state=42)
clf = CalibratedClassifierCV(base_estimator=base_svc, method="sigmoid", cv=3)

pipeline = Pipeline([
    ("tfidf", TfidfVectorizer(
        lowercase=True,
        strip_accents="unicode",
        analyzer="word",
        ngram_range=(1, 2),
        max_features=30000,
        min_df=2
    )),
    ("clf", clf),
])

# Train
pipeline.fit(X_train, y_train)
```

Code Snippet 58: Build and Train SVM Pipeline with TF-IDF

Figure 59: SVM Pipeline

For SVM classifier model building, a linear kernel is used for the baseline model as the linear decision boundaries are more sufficient and computationally efficient for this project with high-dimensional sparse text features in argument type classification (GeeksforGeeks, 2025f). The class weights are balanced to ensure the model treats minority classes with equal importance. Then, the SVM is wrapped with a CalibratedClassifierCV using a sigmoid method to allow the model to output better probability estimates but not hard class assignments. The calibration step is important because the probability scores can be used to compare confidence levels across different models and to support interpretability in the debate analyser. The pipeline includes feature extraction with model training, which is well-suited for the training and evaluation process.

4.5.2 CNN

```
# Dataset & Collator
class TextDataset(Dataset):
    def __init__(self, texts: List[str], labels: List[int]):
        self.texts = texts
        self.labels = labels
    def __len__(self): return len(self.texts)
    def __getitem__(self, idx): return self.texts[idx], self.labels[idx]

@dataclass
class Collate:
    tokenizer: AutoTokenizer
    max_len: int
    def __call__(self, batch):
        texts, labels = zip(*batch)
        enc = self.tokenizer(
            list(texts), truncation=True, padding=True,
            max_length=self.max_len, return_tensors="pt"
        )
        return {
            "input_ids": enc["input_ids"],
            "attention_mask": enc["attention_mask"],
            "labels": torch.tensor(labels, dtype=torch.long),
        }
```

Code Snippet 59: Create Dataset Object and Collator

The CNN model requires text and labels to be structured in a way to can be passed into the training process. So, a dataset class is defined to store the debate sentences with the argument type labels. A collator function is also created to handle the tokenisation of each batch, with truncation and padding for the same maximum sequence length and generate attention masks tensors. This process happens dynamically during training and validation as part of the CNN model pipeline with more flexibility and efficiency.

```
# TextCNN Model
class TextCNN(nn.Module):
    def __init__(self, vocab_size, embed_dim, num_classes, filter_sizes=(3,4,5),
num_filters=100, dropout=0.5, pad_id=0):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=pad_id)
        self.convs = nn.ModuleList([
            nn.Conv1d(in_channels=embed_dim, out_channels=num_filters,
kernel_size=k)
            for k in filter_sizes
        ])
        self.dropout = nn.Dropout(dropout)
        self.fc = nn.Linear(num_filters * len(filter_sizes), num_classes)

        for conv in self.convs:
            nn.init.kaiming_normal_(conv.weight, nonlinearity='relu')
            if conv.bias is not None: nn.init.constant_(conv.bias, 0.0)

    def forward(self, input_ids, attention_mask=None):
        x = self.embedding(input_ids)       # (B, T, E)
        x = x.transpose(1, 2)               # (B, E, T)
        conv_outputs = [F.max_pool1d(F.relu(conv(x)), kernel_size=conv(x).shape[-
1]).squeeze(-1) for conv in self.convs]
        h = torch.cat(conv_outputs, dim=1)
        h = self.dropout(h)
        return self.fc(h)
```

Code Snippet 60: Create TextCNN Model

The CNN structure is designed for text and sentence-level classification. The sentences are mapped to a dense embedding vector to let the model learns semantic representations rather than raw IDs. There are parallel 1-D convolution layers with different kernel sizes (3, 4, 5) to scan across the sequence to capture the n-gram patterns. In this model, ReLU activation is used, and the max-over-time pooling can keep the best signal from each filter. The pooled features from all filters are concatenated and passed through dropout to reduce overfitting. It is then fed to a fully connected layer to output the argument type classes. In this model, the convolution weights are initialised with Kaiming normal to help in reducing vanishing and exploding gradients (Matt, 2025).

```
def train_eval_one_time(embed_dim=128, num_filters=100, dropout=0.5, batch_size=32,
lr=1e-3, epochs=5):
    train_ds = TextDataset(X_train, y_train)
    val_ds   = TextDataset(X_val, y_val)
    collate  = Collate(tokenizer, max_len=128)
    train_dl = DataLoader(train_ds, batch_size=batch_size, shuffle=True,
collate_fn=collate)
    val_dl   = DataLoader(val_ds, batch_size=batch_size, shuffle=False,
collate_fn=collate)

    model = TextCNN(len(tokenizer), embed_dim, len(label2id),
num_filters=num_filters, dropout=dropout, pad_id=tokenizer.pad_token_id).to(DEVICE)
    opt = torch.optim.Adam(model.parameters(), lr=lr)
    criterion = nn.CrossEntropyLoss()

    # Training
    for epoch in range(epochs):
        model.train()
        total_loss = 0
        for batch in train_dl:
            opt.zero_grad()
            batch = {k:v.to(DEVICE) for k,v in batch.items()}
            logits = model(batch["input_ids"])
            loss = criterion(logits, batch["labels"])
            loss.backward(); opt.step()
            total_loss += loss.item()
        print(f"Epoch {epoch+1}/{epochs} | Loss: {total_loss/len(train_dl):.4f}")


train_eval_one_time()
```

Code Snippet 61: Train TextCNN model

After that, the TextDataset container, together with Collator, wraps the train and validation datasets and passes them into the TextCNN with the chosen capacity of 128 embedding dimensions, 100 filters per kernel size, and 0.5 dropout. Adam is used as an optimiser, with a learning rate of 0.001 and CrossEntropyLoss. During training, the model shuffles the batches to reduce overfitting and then passes them through the network to produce logits. In this process, the loss is calculated. This model design keeps things simple and efficient by using mini-batches for GPU throughput, dropout for regularisation, and raw logits to pair correctly with cross-entropy.

```
Epoch 1/5 | Loss: 0.7523
Epoch 2/5 | Loss: 0.4997
Epoch 3/5 | Loss: 0.4180
Epoch 4/5 | Loss: 0.3628
Epoch 5/5 | Loss: 0.3211
```

Figure 60: CNN Loss by Epoch

The baseline model training has a failing training loss from 0.75 to 0.32 across 5 epochs, showing that it learns meaningful patterns from the debate sentences and fits the training data in a stable way. The drop each epoch, with smaller gains later, shows the diminishing returns curve as the model nears convergence.

4.5.3 DeBERTa

```
# Metrics
def compute_metrics(pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)
    return {
        "accuracy": accuracy_score(labels, preds),
        "f1_macro": f1_score(labels, preds, average="macro")
    }

# Training Configuration
training_args = TrainingArguments(
    output_dir="logging/deberta/results",
    overwrite_output_dir=True,
    do_train=True,
    do_eval=True,
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="f1_macro",
    greater_is_better=True,

    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    gradient_accumulation_steps=2,

    learning_rate=2e-5,
    weight_decay=0.05,
    num_train_epochs=2,

    fp16=torch.cuda.is_available(),

    dataloader_num_workers=4,
    dataloader_pin_memory=True,
    dataloader_prefetch_factor=2,

    logging_dir="logging/deberta/logs",
    logging_steps=100,
    save_total_limit=2
)
```

Code Snippet 62: Define DeBERTa's Compute Metrics and Training Arguments

For transformer-based model training, a Hugging Face Trainer run is set up to fine-tune DeBERTa-v3-base on Toulmin labels, which use almost the same transfer-learning recipe used at the previous

stage. The compute_metrics function reports accuracy and macro-F1 with predictions taken via argmax on the logits. While the TrainingArguments allows train, evaluate, and checkpoint every epoch, and reload the best checkpoint at the end using macro-F1 as the judge, which is suitable for imbalanced labels. The batch sizes are 8 per device, with 2 gradient accumulation steps, making it a batch of 16. The 0.00002 learning rate and 0.05 weight decay are standard metrics for transformer fine-tuning. Different from transfer learning model setups, there are data loader settings, including workers, pin memory, and prefetch, to improve the throughput. This setup is powerful to be ready for training the full dataset.

```
# Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    tokenizer=tokenizer,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    data_collator=DataCollatorWithPadding(tokenizer),
    compute_metrics=compute_metrics
)

# Train
trainer.train()
```

Code Snippet 63: Create DeBERTa's Trainer Pipeline and Train

[10854/10854 22:05:52, Epoch 2/2]

Epoch	Training Loss	Validation Loss	Accuracy	F1 Macro
1	0.255500	0.254225	0.905192	0.884685
2	0.183900	0.237174	0.925278	0.908011

Figure 61: DeBERTa's Training Summary

The DeBERTa-v3-base model is wrapped into a pipeline, including the TrainingArgument, tokeniser, dataset, padding collator, and compute_metrics function. The model is then started training. This process involves batching, shuffling, forward pass, loss, backprop, optimiser steps, and gradient accumulation. The learning table shows both training and validation loss fall from epoch 1 to 2, while accuracy rises from 0.905 to 0.925, and macro-F1 improves from 0.885 to 0.908.

4.5.4 GPT

```
# Metrics
def compute_metrics(pred):
    labels = pred.label_ids
    preds = np.argmax(pred.predictions, axis=1)
    return {
        "accuracy": accuracy_score(labels, preds),
        "f1_macro": f1_score(labels, preds, average="macro")
    }

# Training arguments
training_args = TrainingArguments(
    output_dir="model/gpt2/results",
    overwrite_output_dir=True,
    do_train=True,
    do_eval=True,
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="f1_macro",
    greater_is_better=True,

    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    gradient_accumulation_steps=2,

    learning_rate=2e-5,
    weight_decay=0.05,
    num_train_epochs=2,

    fp16=torch.cuda.is_available(),

    dataloader_num_workers=4,
    dataloader_pin_memory=True,
    dataloader_prefetch_factor=2,

    logging_dir="model/gpt2/logs",
    logging_steps=100,
    save_total_limit=2
)
```

Code Snippet 64: Define GPT's Compute Metrics and Training Arguments

The fine-tuning of the GPT-2 transformer for Toulmin label classification also uses Hugging Face Trainer, with metrics and parameters similar to the DeBERTa run, so both transformer models are compared fairly. It also has a compute_metrics function and TrainingArguments that mirror the earlier model recipe, to ensure the performance differences reflect the model architecture rather than the training schedule.

```
# Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    tokenizer=tokenizer,
    train_dataset=train_ds,
    eval_dataset=val_ds,
    data_collator=DataCollatorWithPadding(tokenizer),
    compute_metrics=compute_metrics
)

# Train
trainer.train()
```

Code Snippet 65: Create GPT's Trainer Pipeline and Train

[10854/10854 1:22:14, Epoch 2/2]

Epoch	Training Loss	Validation Loss	Accuracy	F1 Macro
1	0.337200	0.319536	0.875386	0.851450
2	0.321100	0.289336	0.891325	0.866467

Figure 62: GPT's Training Summary

After the pipeline is set and training starts, the learning summary table shows that the GPT-2 baseline model has a small training loss that falls from 0.337 to 0.321, validation loss from 0.320 to 0.289, while accuracy rises from 0.876 to 0.891, and macro-F1 improves from 0.851 to 0.866 from epoch 1 to 2. This indicates a good generalisation and more balanced performance across classes, including C, G, W/Q, and OTH. In a direct comparison on accuracy and macro-F1, GPT-2 lands a bit lower than DeBERTa-v3.

4.5.5 DeepSeek

The classification model used for the DeepSeek model is named DeepSeek-R1-Distill-Qwen-1.5 B, which is implemented using a memory-efficient, quantisation plus low-rank adapter (QLoRA) framework to enable DeepSeek to be fine-tuned on modest hardware without sacrificing much accuracy. QLoRA is an efficient fine-tuning method that initialises the backbone at 4-bit precision (doubly quantised NF4) and trains tiny low-rank adapters with frozen quantised weights, achieving performance almost close to full-precision fine-tuning but with much less memory usage (Dettmers et al., 2023; Ding et al., 2023).

```
# QLoRA config
bf16_ok = torch.cuda.is_available() and
torch.cuda.get_device_capability(0)[0] >= 8
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_compute_dtype=torch.bfloat16 if bf16_ok else
torch.float16,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
)
OFFLOAD_DIR = os.path.join(OUTPUT_DIR, "offload")
os.makedirs(OFFLOAD_DIR, exist_ok=True)
```

Code Snippet 66: Initiate Quantisation for DeepSeek

Firstly, the model is loaded in 4-bit with normalised float 4-bit data types quantisation. The math is set to be done in bfloat16 to keep computation stable.

```
# Model
base_model = AutoModelForSequenceClassification.from_pretrained(
    MODEL_CKPT,
    num_labels=len(VALID_LABELS),
    quantization_config=bnb_config,
    device_map="auto",
    trust_remote_code=True,
    offload_folder=OFFLOAD_DIR,
)
base_model.config.pad_token_id = tokenizer.pad_token_id

# LoRA
base_model = prepare_model_for_kbit_training(base_model)
lora_cfg = LoraConfig(
    task_type=TaskType.SEQ_CLS,
    r=8,
    lora_alpha=16,
    lora_dropout=0.05,
    bias="none",
    target_modules=[
        "q_proj", "k_proj", "v_proj", "o_proj",
        "gate_proj", "up_proj", "down_proj"
    ],
)

model = get_peft_model(base_model, lora_cfg)
model.print_trainable_parameters()
```

Code Snippet 67: Initiate LoRA for DeepSeek

With the quantisation configuration, the base model is created, featuring auto device mapping and an offload folder to allow layers to spill gracefully to the CPU when GPU RAM is tight. Then, this network is prepared for k-bit training, which pre-processes the quantised model to avoid the numerical issues during fine-tuning. After that, Parameter-Efficient Fine-Tuning (PEFT) with LoRA is applied, which involves small low-rank adapters with a rank of 8, scaled by alpha 16 and

dropout of 0.05 are inserted to train only these adapters, and the remaining base weights are frozen. The chosen target modules for self-attention include the highly influential parts of each transformer block. The projections for queries, keys, and values will define the matching features using queries to compare with keys, and provide the content (values) to be combined by the attention weights, while the output projection merges the head results back to the model's hidden dimension. The target modules also include the feed-forward (MLP) part, which will expand and compress features, and a gate controls the throughput of the transformed information. Finally, the model with 4-bit quantisation and LoRA configuration is wrapped together, forming QLoRA, which enables DeepSeek to be adapted to the argument type classification training task on available hardware with quality comparable to full fine-tuning.

```
trainable params: 9,238,528 || all params: 1,552,958,976 || trainable%: 0.5949
```

Figure 63: DeepSeek's Trainable Parameters After QLoRA

The PEFT-LoRA stats show that only about 0.5949% of the 1.55B base weights are trainable parameters. This reduces much GPU memory usage for optimiser states and gradients, and gives faster training and lower overfitting risk, and at the same time, preserves DeepSeek's general language knowledge.

```
# Data collator
data_collator = DataCollatorWithPadding(tokenizer=tokenizer,
pad_to_multiple_of=8)

# Trainer
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=-1)
    return {
        "accuracy": accuracy_score(labels, preds),
        "f1_macro": f1_score(labels, preds, average="macro")
    }

training_args = TrainingArguments(
    output_dir="logging/deepseek-lora/results",
    overwrite_output_dir=True,
    do_train=True,
    do_eval=True,
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="f1_macro",
    greater_is_better=True,

    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    gradient_accumulation_steps=2,

    learning_rate=2e-5,
    weight_decay=0.05,
    num_train_epochs=2,

    fp16=torch.cuda.is_available(),

    dataloader_num_workers=4,
    dataloader_pin_memory=True,
    dataloader_prefetch_factor=2,

    logging_dir="logging/deepseek-lora/logs",
    logging_steps=100,
    save_total_limit=2
)
```

Code Snippet 68: Define DeepSeek's Compute Metrics and Training Arguments

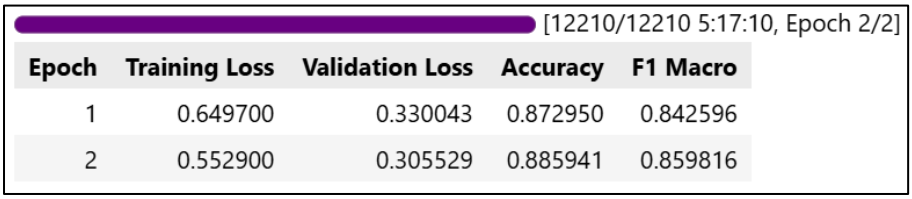
The QLoRA-adapted DeepSeek model also has the same training schedule used for DeBERTa and GPT to ensure a fair, apples-to-apples comparison across transformer baselines. The compute metrics function is also created for this model training, similar to other transformer model building, which reports accuracy and macro-F1. The TrainingArguments mirror the earlier runs, including evaluation and checkpoint during each epoch, small per-device batches, learning rate and weight

decay, and two epochs for a solid baseline. In the QLoRA context, these arguments drive the same training loop, but only a small LoRA adapters receive gradients, while the 4-bit base remains frozen.

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_ds,
    eval_dataset=val_ds,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)

# Train
trainer.train()
```

Code Snippet 69: Create DeepSeek's Trainer Pipeline and Train



Epoch	Training Loss	Validation Loss	Accuracy	F1 Macro
1	0.649700	0.330043	0.872950	0.842596
2	0.552900	0.305529	0.885941	0.859816

Figure 64: DeepSeek's Training Summary

Then, the Trainer collects the model, training arguments, train and validation dataset, tokeniser, data collator, and compute metrics function to start training. The run summary shows the training loss drops from 0.650 to 0.553 and validation loss from 0.330 to 0.306, while accuracy rises from 0.873 to 0.886, and macro-F1 improves from 0.843 to 0.860 from epoch 1 to 2. DeepSeek-QLoRA lands slightly below GPT-2 and below DeBERTa, but has competitive macro-F1 with a tiny fraction of trainable parameters.

4.5.6 LLaMA

The model used for training argument type classification is TinyLlama-1.1B-Chat, together with QLoRa, with the same configuration as DeepSeek, so the results are also comparable and give a lighter training burden to the computer.

```
# QLoRA (4-bit) + LoRA
# 4-bit quantization config for QLoRA
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
)

# Load base model quantized in 4-bit
base_model = AutoModelForSequenceClassification.from_pretrained(
    model_ckpt,
    num_labels=len(VALID),
    quantization_config=bnb_config,
    device_map="auto",
    trust_remote_code=True,
)
base_model.config.pad_token_id = tokenizer.pad_token_id

# Prepare for k-bit training + LoRA
base_model = prepare_model_for_kbit_training(base_model)

lora_cfg = LoraConfig(
    task_type=TaskType.SEQ_CLS,
    r=8,
    lora_alpha=16,
    lora_dropout=0.05,
    bias="none",

target_modules=["q_proj","k_proj","v_proj","o_proj","gate_proj","up_proj","
down_proj"],
)
model = get_peft_model(base_model, lora_cfg)
model.print_trainable_parameters()
```

Code Snippet 70: Initiate Quantisation and LoRA for TinyLLaMA

To train the TinyLLaMA model, the quantisation is done to load the model in 4-bit, and apply LoRA to make only small adapters trainable, instead of training the whole model's 1.1B base. Both these are combined into the PEFT.

```
trainable params: 6,316,032 || all params: 1,040,836,608 || trainable%: 0.6068
```

Figure 65: TinyLLaMA's Trainable Parameters After QLoRA

Only 6.32 million parameters are trainable out of 1.04 billion, which is about 0.61%. The remaining weights stay frozen in 4-bit so that the optimiser can store gradients and moments only for the small adapters. The 0.61% share is close to the DeepSeek setup because the same LoRA rank and target layers are used, and the difference comes from the size of the model between TinyLlama (1.04B) and DeepSeek (1.55B).

```
data_collator = DataCollatorWithPadding(tokenizer=tokenizer,
pad_to_multiple_of=8)

# Metrics
def compute_metrics(pred):
    from sklearn.metrics import accuracy_score, f1_score
    labels = pred.label_ids
    preds = np.argmax(pred.predictions, axis=1)
    return {"accuracy": accuracy_score(labels, preds), "f1_macro":
f1_score(labels, preds, average="macro")}

# Training
args = TrainingArguments(
    output_dir="model/llama_qlora/results",
    overwrite_output_dir=True,
    do_train=True,
    do_eval=True,
    eval_strategy="epoch",
    save_strategy="no",
    load_best_model_at_end=False,

    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    gradient_accumulation_steps=2,

    learning_rate=2e-5,
    weight_decay=0.05,
    num_train_epochs=2,

    optim="paged_adamw_8bit",

    dataloader_num_workers=4,
    dataloader_pin_memory=True,
    dataloader_prefetch_factor=2,

    logging_dir="model/llama_qlora/logs",
    logging_steps=100,
    save_total_limit=2
)

# Trainer
trainer = Trainer(
    model=model,
    args=args,
    train_dataset=train_ds,
    eval_dataset=val_ds,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)

# Train
trainer.train()
```

Code Snippet 71: Define TinyLLaMA's Compute Metrics, Training Arguments, Pipeline, and Train

The TinyLlama is fine-tuned with the same transformer setup used earlier, but with two small changes, including using a bits and bytes optimiser that stores optimiser states in 8-bit, and skipping checkpointing to reduce disk input output.

[10854/10854 9:19:19, Epoch 2/2]

Epoch	Training Loss	Validation Loss	Accuracy	F1 Macro
1	2.579900	0.324109	0.876491	0.850472
2	2.267000	0.294225	0.888699	0.863847

Figure 66: TinyLLaMA's Training Summary

The training loss drops from 2.5799 to 2.2670 (about 12%), while validation loss also falls from 0.3241 to 0.2942 (about 9%), and both accuracy and macro-F1 improve. There is a scale gap where training loss is much higher than validation loss. It can be understood that due to both losses not being computed under identical conditions, where training loss is a mini-batch average with dropout active, while validation loss is a full dataset average. So, the trend matters more than absolute comparison, and the consistency can be confirmed in the evaluation stage.

4.6 Hyperparameter Tuning

The single baseline model's performance is still insufficient, as hyperparameters need to be tuned to achieve reliable and fair performance across models. The tuning objective is to improve macro-F1 on a held-out validation set. In this project, a simple grid search is used for SVM, while Optuna is employed for the CNN and transformer models, as their parameter spaces are larger and more complex. Optuna's TPE sampler and pruning can stop poor trials early and find good settings with fewer runs (GeeksforGeeks, 2025d). To manage costs for transformers and speed up the process, tuning is first done on a stratified subset of 100 debate transcripts. Then, the best configuration is retrained on the full training set and evaluated on the same validation split.

4.6.1 SVM

```
# Hyperparameter tuning (GridSearchCV)
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import make_scorer, f1_score, accuracy_score

# Define param grid for the existing pipeline steps
param_grid = {
    # TF-IDF
    "tfidf__ngram_range": [(1,1), (1,2)],
    "tfidf__max_features": [20000, 30000, 50000],
    "tfidf__min_df": [1, 2, 3],

    # LinearSVC (inside CalibratedClassifierCV)
    "clf__base_estimator__C": [0.5, 1.0, 2.0],
    "clf__base_estimator__class_weight": [None, "balanced"],

    # Calibration method (sigmoid usually fine; isotonic can be better but slower)
    "clf__method": ["sigmoid", "isotonic"],
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scoring = {
    "f1_macro": make_scorer(f1_score, average="macro"),
    "accuracy": make_scorer(accuracy_score),
}
```

Code Snippet 72: Set Up Hyperparameter for SVM

For hyperparameter tuning on SVM, a 5-fold stratified GridSearchCV is run, jointly searching TF-IDF settings on gram numbers, vocabulary size, cutoff, and the LinearSVC's regularisation strength and probability calibration method. The chosen range takes meaning, such as bi-grams that can capture short argumentative phrases, min_df filters noise from rare tokens, 'C' balances between margin size and overfitting, class weight is tested to counter skewed labels, and CalibratedClassifierCV can add probabilities to Linear SVC, which is for thresholding, ensembling, or ranking. Plus, with the stratification, it keeps each fold's argument type label mix consistent.

```
grid = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    scoring=scoring,
    refit="f1_macro",
    cv=cv,
    n_jobs=-1,
    verbose=1,
    error_score="raise"
)

grid.fit(X_train, y_train)
```

Code Snippet 73: Implement Hyperparameter Tuning on SVM

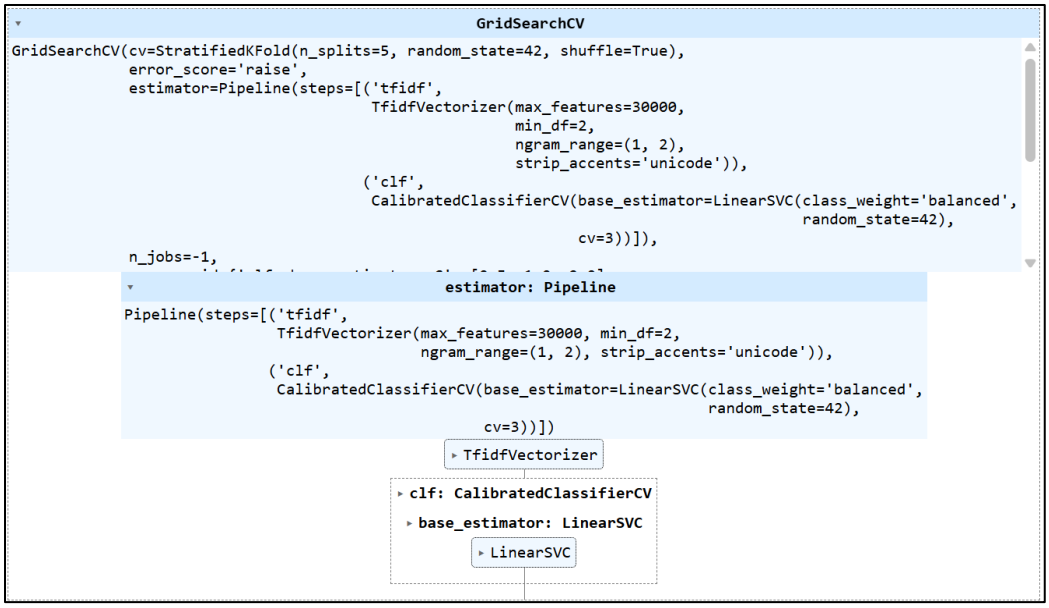


Figure 67: SVM Grid Search Pipeline

Once the parameters are set, the GridSearchCV takes the whole TD-IDF and Calibrated LinearSVC pipeline and tries every hyperparameter combination, scoring with both accuracy and macro-F1 under the 5-fold stratified split. The pipeline is ready for hyperparameter tuning, which includes training, cross-validating, selecting, and refitting.

```
=== Best Params (by f1_macro) ===

{'clf__base_estimator__C': 0.5, 'clf__base_estimator__class_weight': None,
'clf__method': 'isotonic', 'tfidf__max_features': 50000, 'tfidf__min_df': 1,
'tfidf__ngram_range': (1, 2)}

Best CV f1_macro: 0.7702
```

Figure 68: Best Hyperparameters for SVM

The results show the grid search found a regularised SVM with the TF-IDF setup for balance argument type classification. 0.5 ‘C’ shows that a wider margin generalises better than a tighter fit, and no class weighting outperforms balanced. Isotonic calibration is preferred over sigmoid, as it produces non-parametric mapping that yields better macro-F1 probability estimates. The search chose a larger vocabulary (50k features), kept rare terms (min_df=1), and uni+bi-grams (1,2), which capture specific argumentative phrases. The best cross-validated macro-F1 = 0.7702 shows balanced performance across four argument type classes. The baseline and best-tuned models will be evaluated later.

4.6.2 CNN

```
# Hyperparameter Tuning
def objective(trial):
    embed_dim   = trial.suggest_categorical("embed_dim", [64,128,256])
    num_filters = trial.suggest_categorical("num_filters", [50,100,200])
    dropout     = trial.suggest_float("dropout", 0.3, 0.7)
    batch_size  = trial.suggest_categorical("batch_size", [16,32,64])
    lr          = trial.suggest_loguniform("lr", 1e-4, 1e-2)
    epochs      = 3

    # Same training loop but return val accuracy
    train_ds = TextDataset(X_train, y_train)
    val_ds   = TextDataset(X_val, y_val)
    collate  = Collate(tokenizer, max_len=128)
    train_dl = DataLoader(train_ds, batch_size=batch_size, shuffle=True,
collate_fn=collate)
    val_dl   = DataLoader(val_ds, batch_size=batch_size, shuffle=False,
collate_fn=collate)

    model = TextCNN(len(tokenizer), embed_dim, len(label2id),
num_filters=num_filters, dropout=dropout,
pad_id=tokenizer.pad_token_id).to(DEVICE)
    opt = torch.optim.Adam(model.parameters(), lr=lr)
    criterion = nn.CrossEntropyLoss()

    for epoch in range(epochs):
        model.train()
        for batch in train_dl:
            opt.zero_grad()
            batch = {k:v.to(DEVICE) for k,v in batch.items()}
            logits = model(batch["input_ids"])
            loss = criterion(logits, batch["labels"])
            loss.backward(); opt.step()

    # Eval
    model.eval()
    all_preds, all_labels = [], []
    with torch.no_grad():
        for batch in val_dl:
            batch = {k:v.to(DEVICE) for k,v in batch.items()}
            preds = model(batch["input_ids"]).argmax(dim=-1).cpu().tolist()
            all_preds.extend(preds);
all_labels.extend(batch["labels"].cpu().tolist())

    return accuracy_score(all_labels, all_preds)

# Run tuning
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=20)
print("Best params:", study.best_params)
```

Code Snippet 74: Implement Hyperparameter Tuning on TextCNN

Optuna is used to automatically tune the CNN's key knobs and pick the setting with the best scores on the validation split. The search space is defined, including the embedding size, number of convolution filters, dropout, batch size, and learning rate on a log scale. Then, they are run in a

training loop with 3 epochs with Adam and cross-entropy on tokenised and padded batches. After each training, the model is switched to evaluation mode and returns the validation accuracy, which Optune will maximise over 20 trials using its sampler. After finishing, the best configuration is used to retrain on the full schedule.

```
Best params: {'embed_dim': 256, 'num_filters': 100, 'dropout':
0.35397716592326217, 'batch_size': 32, 'lr': 0.0008308143957401575}
```

Figure 69: Best Hyperparameters for TextCNN

For the best tuning parameters of CNN, Optuna picks a larger embedding size of 256, and 100 filter count per kernel size. The dropout of about 0.354 is to curb overfitting, batch size 32 balances gradient stability, and the learning rate of about 0.00083 is a bit lower than the baseline.

4.6.3 DeBERTa, GPT, DeepSeek, and LLaMA

For hyperparameter tuning of transformer models, a smaller slice of the corpus is used, so searches run with a lesser computing burden across DeBERTa, GPT-2, DeepSeek-QLoRA, and TinyLlama-QLoRA. It explores learning rate, batch size, dropout/LoRA settings, and more to find the best configuration by macro-F1 score, then re-trains on the full training data for final evaluation.

```
unique_files = df[filename_col].dropna().unique().tolist()
rng = np.random.default_rng(42)
sampled_files = rng.choice(unique_files, size=min(100, len(unique_files)),
replace=False).tolist()
df_sample = df[df[filename_col].isin(sampled_files)].copy()

train_texts_s, val_texts_s, train_labels_s, val_labels_s = train_test_split(
    df_sample['input'].tolist(),
    df_sample['label'].tolist(),
    test_size=0.2,
    stratify=df_sample['label'],
    random_state=42
)

def make_hf_datasets(tokenizer, texts_train, labels_train, texts_val,
labels_val, max_length=256):
    enc_tr = tokenizer(texts_train, truncation=True, padding=True,
max_length=max_length)
    enc_va = tokenizer(texts_val,   truncation=True, padding=True,
max_length=max_length)
    ds_train = Dataset.from_dict({
        "input_ids": enc_tr["input_ids"],
        "attention_mask": enc_tr["attention_mask"],
        "labels": labels_train
    })
    ds_val = Dataset.from_dict({
        "input_ids": enc_va["input_ids"],
        "attention_mask": enc_va["attention_mask"],
        "labels": labels_val
    })
    return ds_train, ds_val
```

Code Snippet 75: Sample 100 Unique Debate Transcripts

Firstly, 100 unique debates are randomly sampled using a fixed seed for reproducibility, and an 80/20 stratified split is created to have a balanced distribution across all argument types in both train and validation. The dataset texts are then tokenised with truncation or padding with a maximum length of 256. This setup standardises the tuning pipeline across all four transformers.

```
def objective(trial):
    # compact search space for quick tuning
    lr = trial.suggest_float("learning_rate", 1e-5, 5e-4, log=True)
    weight_decay = trial.suggest_float("weight_decay", 0.0, 0.1)
    batch_size = trial.suggest_categorical("per_device_train_batch_size",
[8, 16, 32])
    grad_accum = trial.suggest_categorical("gradient_accumulation_steps",
[1, 2, 4])
    max_length = trial.suggest_categorical("max_length", [128, 256, 384])
    warmup_ratio = trial.suggest_float("warmup_ratio", 0.0, 0.15)
    num_epochs = trial.suggest_categorical("num_train_epochs", [1, 2])

    ds_train, ds_val = make_hf_datasets(
        tokenizer,
        train_texts_s, train_labels_s,
        val_texts_s,   val_labels_s,
        max_length=max_length
    )

    model_hpo = AutoModelForSequenceClassification.from_pretrained(
        model_ckpt, num_labels=len(valid), use_safetensors=True
    )
```

Code Snippet 76: Set Up Hyperparameters for Model without LoRA

```
def objective(trial):
    lr           = trial.suggest_float("learning_rate", 5e-6, 5e-4, log=True)
    weight_decay = trial.suggest_float("weight_decay", 0.0, 0.1)
    warmup_ratio = trial.suggest_float("warmup_ratio", 0.0, 0.15)
    batch_size   = trial.suggest_categorical("per_device_train_batch_size", [2, 4])
    grad_accum   = trial.suggest_categorical("gradient_accumulation_steps", [4, 8])
    max_len      = trial.suggest_categorical("max_length", [64, 96, 128])
    num_epochs   = trial.suggest_categorical("num_train_epochs", [1, 2])

    # LoRA params
    lora_r       = trial.suggest_categorical("lora_r", [4, 8, 16])
    lora_alpha   = trial.suggest_categorical("lora_alpha", [8, 16, 32])
    lora_dropout = trial.suggest_float("lora_dropout", 0.0, 0.1)

    tr_ds, va_ds = make_hf_ds(train_s, val_s, max_length=max_len)

    base = AutoModelForSequenceClassification.from_pretrained(
        model_ckpt,
        num_labels=len(VALID),
        quantization_config=bnb_config,
        device_map="auto",
        trust_remote_code=True,
    )
    base.config.pad_token_id = tokenizer.pad_token_id
    base = prepare_model_for_kbit_training(base)

    lcfg = LoraConfig(
        task_type=TaskType.SEQ_CLS,
        r=lora_r, lora_alpha=lora_alpha, lora_dropout=lora_dropout,
        bias="none",

target_modules=["q_proj","k_proj","v_proj","o_proj","gate_proj","up_proj","down_proj"],
    )
    model_hpo = get_peft_model(base, lcfg)
```

Code Snippet 77: Set Up Hyperparameters for Model with LoRA

Then, the Optuna setup runs a lightweight hyperparameter search for the transformer classifier on a 100-debate subset, aiming to maximise macro-F1 on the validation split, like hyperparameter tuning in CNN. The trial samples a mix of knobs, including learning rate, weight decay, per-device batch size, gradient accumulation, sequence length, warmup ratio, and a short epoch count. For DeepSeek and TinyLLaMA that run QLoRa, the LoRA rank, alpha, and dropout are the hyperparameters that can be examined. Then, the model is loaded according to their baseline model, including DeBERTa-v3, GPT-2, DeepSeek-R1, and TinyLLaMA.

```
    args = TrainingArguments(
        output_dir=f"logging/deberta/hpo_trial_{trial.number}",
        overwrite_output_dir=True,
        do_train=True,
        do_eval=True,
        eval_strategy="epoch",
        save_strategy="epoch",
        load_best_model_at_end=True,
        metric_for_best_model="f1_macro",
        greater_is_better=True,

        per_device_train_batch_size=batch_size,
        per_device_eval_batch_size=min(batch_size, 32),
        gradient_accumulation_steps=grad_accum,

        learning_rate=lr,
        weight_decay=weight_decay,
        num_train_epochs=num_epochs,
        warmup_ratio=warmup_ratio,

        fp16=torch.cuda.is_available(),
        logging_steps=50,
        save_total_limit=1,
        report_to=[],
        dataloader_num_workers=2,
        dataloader_pin_memory=True
    )
```

Code Snippet 78: Set Up Training Arguments for Transformers

```
    trainer_hpo = Trainer(
        model=model_hpo,
        args=args,
        tokenizer=tokenizer,
        train_dataset=ds_train,
        eval_dataset=ds_val,
        data_collator=DataCollatorWithPadding(tokenizer),
        compute_metrics=compute_metrics,
        callbacks=[EarlyStoppingCallback(early_stopping_patience=2)]
    )

    trainer_hpo.train()
    eval_res = trainer_hpo.evaluate(ds_val)
    return eval_res["eval_f1_macro"]

study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=10, show_progress_bar=False)
```

Code Snippet 79: Implement Hyperparameter Tuning for Transformer Models

The setup hyperparameters are wrapped into TrainingArguments and inserted into the Trainer pipeline, along with the model, tokeniser, training and validation datasets, data collator, metrics computation initiated during baseline model training, and an early stopping callback to stop unpromising runs. Optuna repeats this for 10 trials, and the best hyperparameter results are used again to train the model in the full dataset.

Model	Best macro-F1	Learning rate	Weight decay	Warmup ratio	Max length	Per-device batch	Grad accum	Epochs	LoRA r	LoRA α	LoRA dropout
DeBERTa-v3	0.759	4.952e-05	0.058	0.104	256	8	1	3	—	—	—
GPT-2	0.753	4.996e-04	0.035	0.054	384	8	1	3	—	—	—
DeepSeek-R1 (QLoRA)	0.760	2.990e-04	0.045	0.108	256	4	1	3	16	32	0.087
TinyLlama (QLoRA)	0.669	1.593e-04	0.040	0.008	96	2	8	3	4	16	0.062

Table 8: Best Hyperparameters of Each Model (Values After Rounding)

The tuned results show the best hyperparameter of each model. DeepSeek-R1 with QLoRA edges out the others (macro-F1 0.760), DeBERTa-v3 is essentially tied (0.759), GPT-2 is close behind (0.753), and TinyLLaMA trails (0.669). DeBERTa prefers a small learning rate (about 5e-5) with relatively strong weight decay (0.058), a larger warmup (0.104), and a 256-token context. GPT-2 works best with a higher learning rate (about 5e-4), moderate weight decay (0.035), a longer context (384 tokens), and the same per-device batch of 8. The QLoRA setups keep memory low by training only adapter weights. DeepSeek runs with 4-bit quantisation plus LoRA rank 16, α 32, dropout 0.087, and uses a 256-token context with a mid-range learning rate (~3e-4), weight decay 0.045, and warmup 0.108. TinyLLaMA uses smaller adapters (rank 4, α 16, dropout 0.062), a short 96-token context, and compensates for the small per-device batch (2) via gradient accumulation (8) to reach an effective batch of 16. However, their actual performance is still unknown until the retraining and final evaluation. These hyperparameters are then sent for full debate dataset training.

4.7 Summary

In summary, Chapter 4 implements the system in detail. This chapter firstly unifies thousands of transcripts, removes duplicates and weak fields, and converts debates to sentences using a two-step approach, including punctuation restoration followed by segmentation, and rule-based discourse cues to fix some cases. This makes the dataset suitable for modelling at the end. Then, a pretrained fallacy classifier is loaded to validate downstream integration. At the same time, a stratified sample of debates is labelled with rationale using the help of ChatGPT into Toulmin argument types, including C→Claim, G→Grounds, W/Q→Warrant/Qualifier, and OTH→Others. A DeBERTa transfer-learning model is trained and then tuned, and achieves validation results of 82% accuracy, with about 0.77 macro-F1, and is used to label the remaining sentences, and 7,060 labels are revised. The chapter then prepares task-specific pipelines for model building. The pipelines include TF-IDF features and a calibrated LinearSVC for the SVM baseline, tokeniser-driven batches for a compact TextCNN, and standardised Hugging Face setups for DeBERTa, GPT-2, DeepSeek-QLoRA, and TinyLLaMA-QLoRA with 4-bit loading and LoRA adapters for memory efficiency. Training is run with consistent arguments (batching, evaluation per epoch, best-model selection, and hyperparameters are tuned fairly. Through hyperparameter tuning, there is a 5-fold grid for SVM and Optuna searches on a 100-debate subset for deep models. The resulting best settings for the transformer model showed a clear pattern, but still need evaluation from full retraining. In short, the project had a fully labelled dataset, reproducible training pipelines, tuned hyperparameters for each model, and baseline and tuned results ready for deeper analysis, comparison, and deployment in the next chapter.

Chapter 5: Results and Discussion

5.1 Introduction

Chapter 5 evaluates the models' performance and explains the learning outcomes from both the baselines and the tuned runs. First, this chapter shows a classification report and confusion matrix for every model. Next, it compares all models side-by-side. Based on these comparisons, the best model will be selected as the main debate sentence argument types. Finally, the chapter outlines deployment with a Streamlit app that enables users to input a debate transcript, run sentence-level argument type classification, apply the fallacy detector, and generate a recommendation, aligning the process with the project objectives.

5.2 Dataset Corpus Evaluation

Before model evaluations, note that the transfer learning model completes the dataset labelling used for training the model. So, it is important to estimate the true accuracy of the final labelled corpus by quantifying the quality of labels before treating downstream scores as definitive (not looking at the final model's accuracy only as the result). The estimated accuracy of the final labelled dataset can be computed using the class-wise precision from the transfer learning classification report (Manning et al., 2009; Northcutt et al., 2021). Based on this project, let the set of classes be

$$K = \{C, G, W/Q, OTH\}$$

For each class $k \in K$:

- $S_k = Number\ of\ "seed"\ (sampling\ labels)$,
- $P_k = Total\ predicted\ labels\ using\ transfer\ learning\ model\ excluding\ seed$,
- $R_k = Number\ of\ revised\ predicted\ labels$,
- $U_k = P_k - R_k\ (Number\ of\ unrevised\ predicted\ labels)$.

Let $\hat{\pi}_k$ be the precision for class k (which is measured by $\hat{\pi}_k = \frac{TP_k}{TP_k + FP_k}$), and it is the fraction of predicted-as-k items that are actually k, so the expected number of correct labels is

$$\hat{C}_k = S_k + R_k + \hat{\pi}_k U_k$$

This is because S_k and R_k are part of the corpus that was manually corrected, they can be claimed as reliable by the annotator who checked (Krippendorff, 2011). And with this, the dataset corpus accuracy can be estimated with the weighted average

$$\widehat{Acc} = \frac{\sum_{k \in K} \hat{C}_k}{\sum_{k \in K} (S_k + R_k + U_k)} = \frac{\sum_{k \in K} (S_k + R_k + \hat{\pi}_k U_k)}{N}$$

Where N is the total number of sentences. This is a direct plug-in of the standard precision definition from information retrieval (Manning et al., 2009).

k	S_k	P_k	R_k	U_k	$\hat{\pi}_k$ (from TL report)	$\hat{C}_k$
C	922	16,018	177	15,841	0.88	15,039.08
G	1,272	27,408	392	27,016	0.83	24,087.28
W/Q	1,695	47,454	5,961	41,493	0.85	42,925.05
OTH	418	13,345	530	12,815	0.51	7,483.65
Total	4,307	104,225	7,060	97,165	—	**89,535.06**

Table 9: Argument Type Distribution in Revised Processed Dataset

Hence, the revised labelled dataset corpus accuracy is estimated as

$$\widehat{Acc} = \frac{89{,}535.06}{108{,}532} \approx 82.5\%$$

The estimated label accuracy is ≈ 82.5%. In all the following downstream results, the observed model scores should be reported separately from this dataset quality estimate, since evaluation on noisy labels can be optimistic or pessimistic, unless an audit is performed to create a small gold-standard test set with tighter confidence intervals (Thompson, n.d.). Still, in this project, the gold audit is not feasible within time, knowledge, and resource limits. Therefore, this chapter presents the dataset label quality estimate and final model scores on current labels, and provides an analysis to interpret the results and demonstrate how conclusions can differ.

5.3 Model Evaluations and Discussions

The performance of both the baseline and tuned models is evaluated and discussed using the same method, allowing for easy comparison. For each model, predictions are made on the full validation split, and then three views are printed.

```
# Run predictions
preds = trainer.predict(val_ds)
y_pred = np.argmax(preds.predictions, axis=1)

# Classification Report
print("\n=== Classification Report ===")
labels_present = sorted(set(val_labels))
target_names_present = [inv_label_map[i] for i in labels_present]

print(classification_report(
    val_labels,
    y_pred,
    labels=labels_present,
    target_names=target_names_present,
    zero_division=0
))
```

Code Snippet 80: Print Classification Report

```
# Confusion Matrix
y_true = val_labels
cm = confusion_matrix(y_true, y_pred)
print("Confusion Matrix:\n", cm)

# Plot as heatmap
# ['C', 'G', 'W/Q', 'OTH']
labels_order = [inv_label_map[i] for i in range(len(inv_label_map))]

disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=labels_order)
disp.plot(cmap=plt.cm.Blues, xticks_rotation=45)
plt.title("Confusion Matrix")
plt.tight_layout()
plt.show()
```

Code Snippet 81: Plot Confusion Matrix

```
# Compute per-class ROC and AUC
fprs, tprs, aucs = {}, {}, {}
for j, cls_id in enumerate(labels_present):
    # Skip if no positive samples for this class in y_true
    if y_true_bin[:, j].sum() == 0:
        continue
    fpr, tpr, _ = roc_curve(y_true_bin[:, j], probs[:, j])
    fprs[cls_id], tprs[cls_id] = fpr, tpr
    aucs[cls_id] = auc(fpr, tpr)

# Micro-average ROC/AUC
fpr_micro, tpr_micro, _ = roc_curve(y_true_bin.ravel(),
probs.ravel())
auc_micro = auc(fpr_micro, tpr_micro)

# Macro-average AUC (mean across available per-class AUCs)
auc_macro = float(np.mean(list(aucs.values()))) if len(aucs) > 0
else float("nan")

# Plot ROC curves
import matplotlib.pyplot as plt
plt.figure(figsize=(7, 6))
# Per-class curves
for cls_id in labels_present:
    if cls_id in fprs:
        plt.plot(fprs[cls_id], tprs[cls_id], lw=2,
                 label=f"{inv_label_map[cls_id]}
(AUC={aucs[cls_id]:.3f})")

# Micro-average
plt.plot(fpr_micro, tpr_micro, lw=2, linestyle="--",
         label=f"micro-average (AUC={auc_micro:.3f})", alpha=0.9)

# Chance line
plt.plot([0, 1], [0, 1], color="gray", lw=1, linestyle=":")

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves — One-vs-Rest")
plt.legend(loc="lower right")
plt.tight_layout()
plt.show()
```

Code Snippet 82: Plot ROC Curves

Firstly, a classification report, including precision, recall, F1, support, and macro-F1, is shown to reflect class balance across argument types. Then, there is a confusion matrix to visualise where errors concentrate, which uses the same label order as the accurate classification set. Lastly, the ROC curves are computed from softmax probabilities to assess ranking performance independent of any fixed threshold.

5.3.1 SVM

```
=== Classification Report ===
              precision    recall  f1-score   support

           C       0.86      0.73      0.79      3388
           G       0.83      0.82      0.82      5736
         OTH       0.71      0.56      0.63      2753
         W/Q       0.82      0.92      0.87      9830

    accuracy                           0.82     21707
   macro avg       0.80      0.76      0.78     21707
weighted avg       0.81      0.82      0.81     21707
```

Figure 70: SVM Classification Report

From an overall view, the baseline SVM is uneven across classes. The overall accuracy is 0.82, and macro-F1 is 0.78. This indicates that dominant classes are enhancing performance. W/Q is the largest class with 9,830 supports, showing a very high recall of 0.92 with a precision of 0.82. This means that for every 100 true W/Q sentences, about 92 are caught, but this aggressiveness also pulls in non-W/Q cases that inflate false positives. G is balanced from precision, recall, and F1-score, which are about 0.83, 0.82, and 0.82 on 5,736 items, respectively. C has high precision (0.86) but lower recall (0.73), meaning the classifier is conservative about calling something a Claim. Out of 100 true Claims, there are 27 are missed, and this can be analysed that it drift into W/Q or G when explicit claim markers are weak. The weakest area is OTH with a precision of 0.71, a recall of 0.56, and an F1-score of 0.63 on 2,753 items. This low recall implies that nearly 44% of true OTH sentences are classified incorrectly, indicating a problem with lexically diffuse sentences. The gap between accuracy (0.82) and macro-recall (0.76) shows that the model classifies the common classes well, but under-recovers minority or fuzzier categories. In short, the baseline SVM is good at W/Q and G detection, a bit cautious on C, and struggles most with OTH.

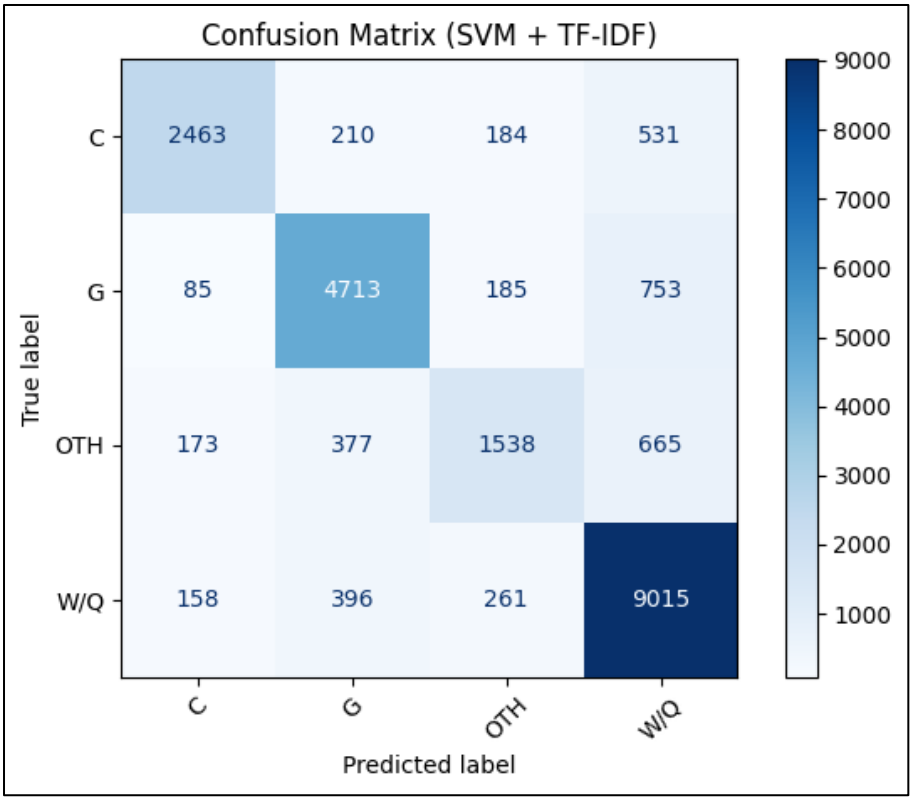


Figure 71: SVM Confusion Matrix

The confusion matrix shows that the model gets many classifications right (as presented by 82% accuracy). The diagonal cells are the correct hits. For example, C has 2,463, G has 4,713, OTH has 1,538, and W/Q has 9,015 correct. For true C with 73% recall, there are 531 cases, which is about 16% of all C cases that are mistaken as W/Q, with a smaller proportion to G (6%) and OTH (5%). True G is stable a bit at roughly 82% recall, but with 735 sentences still drifting into W/Q. The most difficult class is OTH. Only about 56% are kept as OTH, while 665 move to W/Q and 377 move to G, which explains the low OTH recall and F1-score. Most of the sentences are classified as W/Q. There are a total of 1,949 items by adding up the off-diagonal entries landing in the W/Q column from C, G, and OTH. That volume explains why W/Q recall is very high, and the precision is nearly 82%.

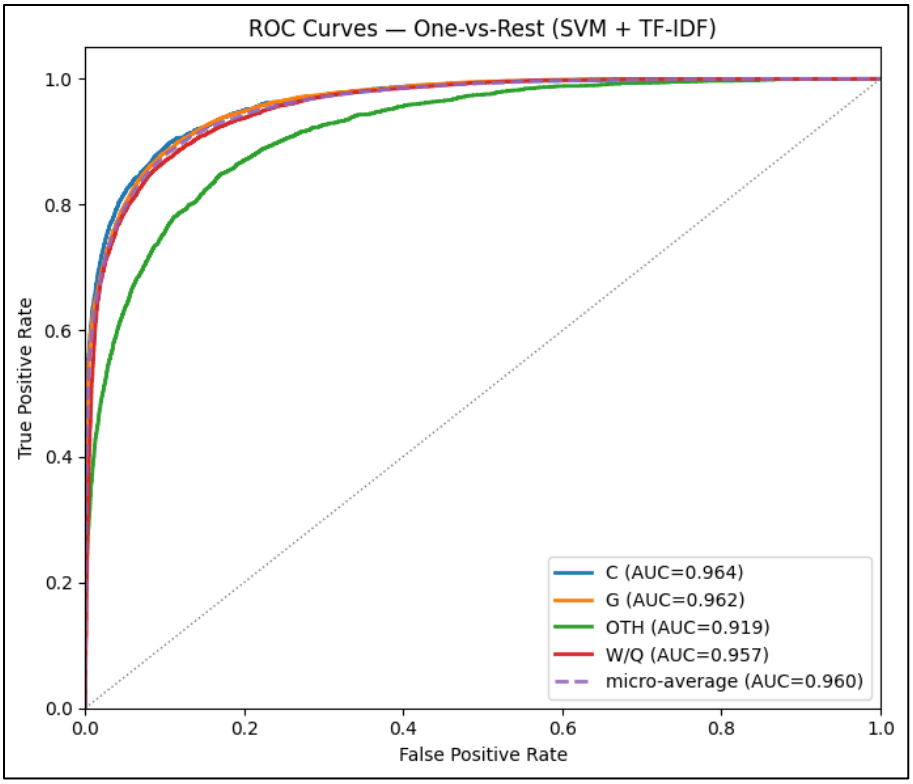


Figure 72: SVM ROC Chart

The ROC chart shows how well the SVM can separate each class from the others across all possible score cutoffs. Curves that stay near the top-left corner mean the model can keep a high true-positive rate. All classes have strong area-under-curve scores. C and G are highest at about 0.964 and 0.962, while W/Q is close behind at about 0.957, matching its strong recall from earlier. OTH is the hardest, with an AUC of 0.919. The micro-average AUC is about 0.960, which shows strong overall ranking performance across all sentences. These curves suggest that tuning class-specific thresholds can trade a little precision to recover more C or tighten the W/Q threshold to reduce the spill-over from OTH, without too significant losses. From the ROC chart overall, the model ranks classes well, and threshold tuning offers room to balance errors by class.

```
=== Classification Report (Best Model) ===
              precision    recall  f1-score   support

           C       0.85      0.73      0.78      3388
           G       0.82      0.83      0.83      5736
         OTH       0.72      0.57      0.64      2753
         W/Q       0.83      0.91      0.87      9830

    accuracy                           0.82     21707
   macro avg       0.81      0.76      0.78     21707
weighted avg       0.82      0.82      0.82     21707
```

Figure 73: SVM Classification Report (Best Tuned)

The tuned SVM looks almost the same as the baseline. Overall accuracy stays at 0.82, and macro-F1-score stays at 0.78. By class, C changes a little in precision (0.86 to 0.85) while recall is unchanged (0.73), so its F1 is down from 0.79 to 0.78. G improves recall by a point (0.82 to 0.83) and remains F1 at 0.83. OTH increases by about 1% on both precision and recall (0.71 and 0.56 to 0.72 and 0.57), making the F1 also up from 0.63 to 0.64, which is good but small. W/Q increases 1% of precision (0.82 to 0.83) and drops 1% of recall (0.92 to 0.91), but its F1 is unchanged at 0.87. This means slightly less over-prediction of W/Q. This pattern matches the grid-search choice of isotonic calibration and stronger regularisation, which reduces over-prediction of W/Q and gives a slight gain in weighted scores without affecting macro-F1, although the changes can be said as if no.

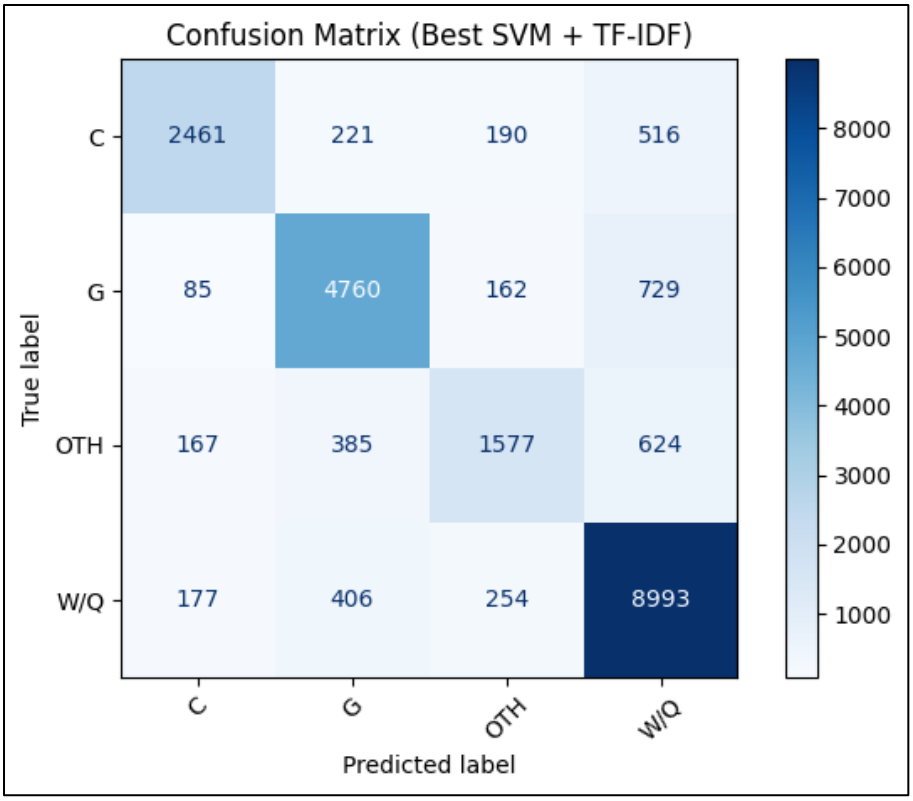


Figure 74: SVM Confusion Matrix (Best Tuned)

Based on the classification report analysis, it can be known that the confusion matrix of the best-tuned SVM model will look similar to the baseline but with small changes in W/Q spill. Indeed, false positives in W/Q drop from 1,949 to 1,869, with fewer C, G, and OTH items being classified there. For C, correct hits are almost unchanged (2,463 to 2,461). Misclassification cases to W/Q are fewer (531 to 516), with small increases going to G and OTH, which is not enough to change the C recall and precision changes only slightly. G improves the most among the big classes; the correct classification rises from 4,713 to 4,760, with fewer mistakes to OTH (185 to 162) and W/Q (753 to 729). The correct OTH grows from 1,538 to 1,577, and the error classification to W/Q falls (665 to 624). W/Q corrects drop from 9,015 to 8,993, and a few more W/Q items move to C or G, matching the small recall drop and the precision gain as shown in the report. Overall, tuning slightly reduces the tendency classification to W/Q, and slightly improves G and OTH in the right direction. The pattern fits a model that is slightly better calibrated and a bit less eager to predict W/Q.

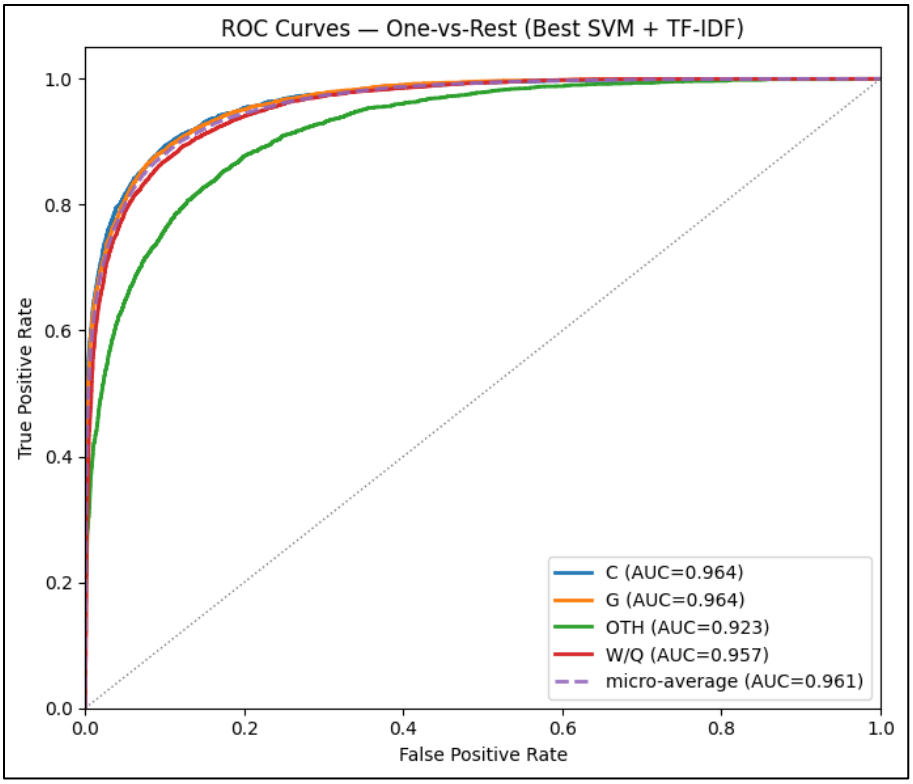


Figure 75: SVM ROC Chart (Best Tuned)

The best-tuned model ROC curves and AUC values present that the SVM's ranking ability is slightly better than the baseline. Claim stays high at AUC 0.964, and Grounds increase from 0.962 to 0.964. W/Q remains at 0.957. OTH improves slightly from 0.919 to 0.923 with the reduced classification to W/Q seen in the confusion matrix. The micro-average rises to 0.961. With AUCs above 0.95 for C, G, and W/Q, ranking for the easier classes is already very strong and remains stable after tuning. Plus, the curves are close to the top-left corner, which means the tuned model can keep true positives high, while keeping weak matches low across many score cutoffs.

Overall, the SVM performs well on W/Q and G but is so-so on C and OTH. The tuned version keeps the same accuracy and macro-F1, and slightly reduces the spill into W/Q, increases G and OTH a little, and improves ranking quality in the ROC curves, with small gains in micro-AUC and OTH AUC. OTH remains the hardest class to separate. In short, tuning stabilises the model and enhances the error classification without changing the overall results (only a small effort).

5.3.2 CNN

Validation Report:

	precision	recall	f1-score	support
C	0.82	0.82	0.82	3388
G	0.89	0.80	0.84	5736
OTH	0.72	0.62	0.67	2753
W/Q	0.85	0.93	0.89	9830
accuracy			0.84	21707
macro avg	0.82	0.79	0.80	21707
weighted avg	0.84	0.84	0.84	21707

Figure 76: CNN Classification Report

The Text CNN shows mostly balanced behaviour on the validation set with 84% accuracy and macro-F1 0.80, which means performance is well-distributed across classes rather than driven only by the largest one. C has a precision of 0.82 and a recall of 0.82, indicating that the model neither over-predicts nor misses many C sentences. G has a high precision of 0.89 but a lower recall of 0.80. This suggests that the CNN is strict about calling G. W/Q, which is the strongest class, with 0.93 recall and 0.85 precision. The model classifies most W/Q sentences well, similar to SVM, which can be due to clear cue phrases and local n-gram patterns that a 1-D convolutional model does well. OTH also remains the hardest, with a precision of 0.72 and a recall of 0.62. The macro-recall 0.79 and weighted-F1 0.84 suggest a good overall fit with gaps focused mainly on OTH and slightly on G.

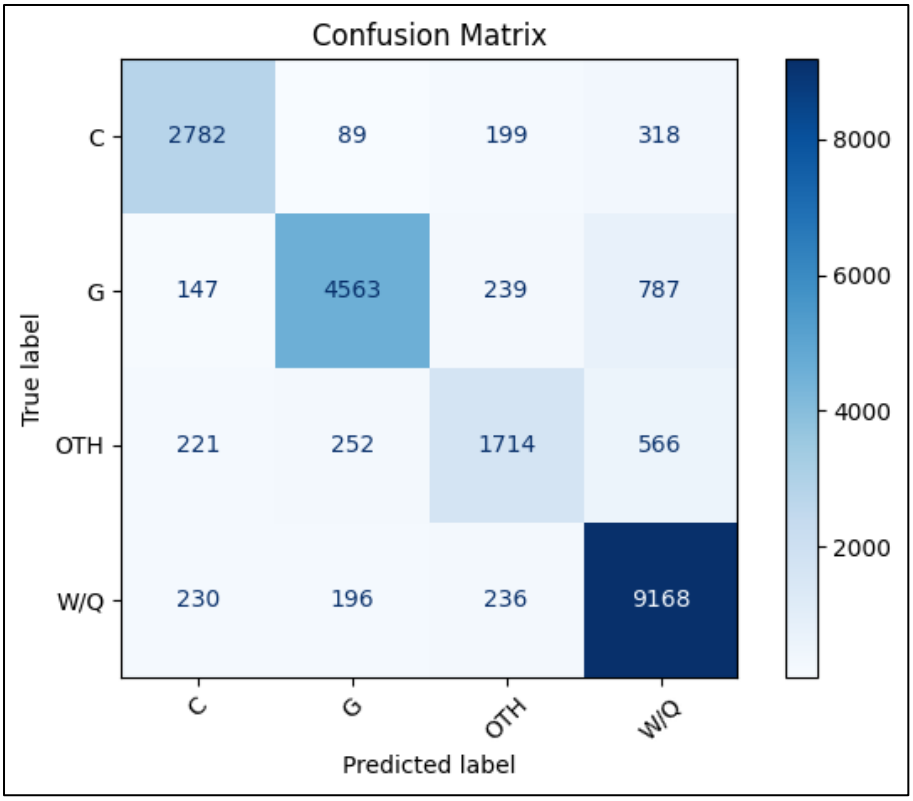


Figure 77: CNN Confusion Matrix

According to the confusion matrix, the CNN baseline hits the diagonal. It exhibits an imbalanced error pattern, as indicated in the classification report, with the model leaning toward W/Q as the default prediction. Recall is strong for W/Q at 9,168 of 9,830 sentences, good for C at 2,782 of 3,388, and for G at 4,563 of 5,736, and weakest for OTH at 1,714 of 2,753. Most mistakes go into W/Q, especially from G with 787 cases and from OTH with 566, suggesting the convolutional features capture lexical cues of reasoning and hedging more easily than evidence or non-argumentative text. C and G are well separated, shown by smaller swaps of 89 and 147, but OTH bleeds into all three argumentative classes. This shows its heterogeneity and the model's limited sentence context. Overall, the matrix indicates a baseline that is dependable for W/Q, normal for C and G, and weak for OTH.

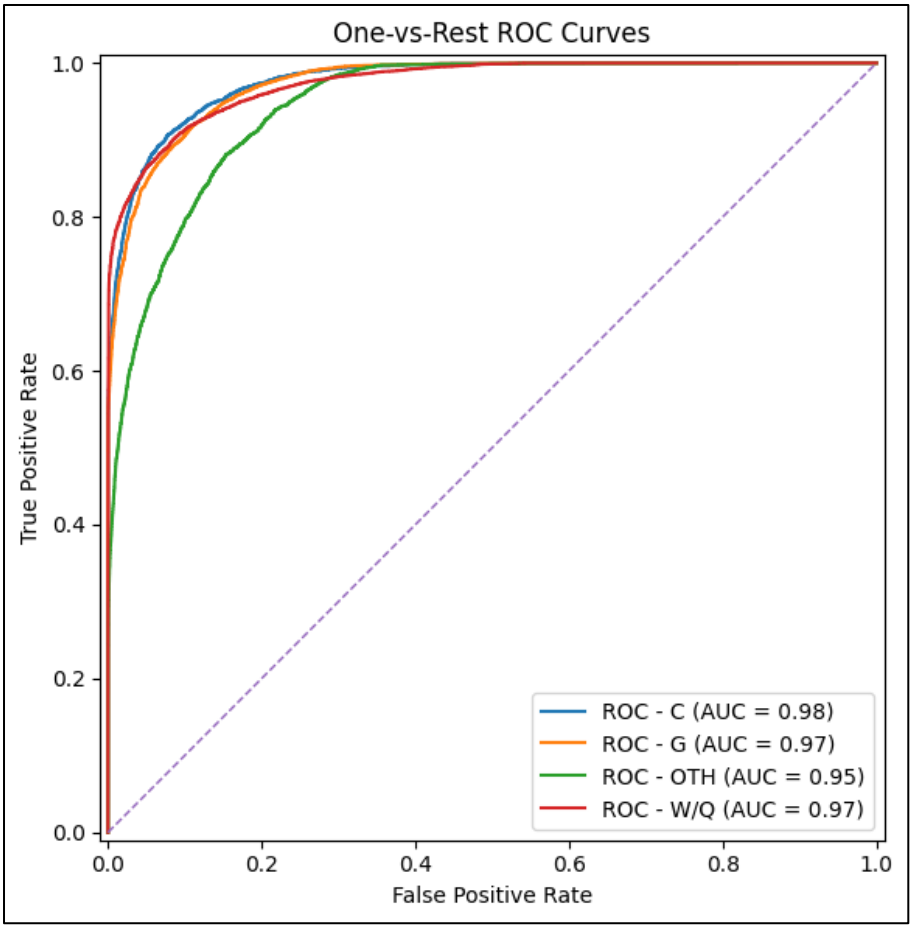


Figure 78: CNN ROC Chart

```
ROC-AUC (OvR, macro):    0.9678
ROC-AUC (OvR, weighted): 0.9710
```

Figure 79: CNN Macro and Weight ROC-AUC

The ROC curves indicate that the CNN performs well in ranking classes across various score cutoffs. C is strongest with an AUC of about 0.98, G and W/Q are close at about 0.97, and OTH is lower but still good at about 0.95. The macro-ROC-AUC is 0.968, which treats every class equally. OTH's lower separability pulls the average down a little. The weighted ROC-AUC is 0.971, which weights classes by how many examples they have, so it is slightly higher because W/Q and G are larger and already easy to separate. The curves that are close to the top-left corner and the values near 1.0 mean the model can keep true positives high while keeping false positives low if the cutoff is chosen well.

```
=== Validation Report (Best Params) ===
              precision    recall  f1-score   support

           C       0.88      0.79      0.83      3388
           G       0.85      0.86      0.85      5736
         OTH       0.71      0.66      0.68      2753
         W/Q       0.88      0.92      0.90      9830

    accuracy                           0.85     21707
   macro avg       0.83      0.81      0.82     21707
weighted avg       0.85      0.85      0.85     21707
```

Figure 80: CNN Classification Report (Best Tuned)

The tuned Text CNN improves overall balance. Accuracy rises from 84% to 85% and macro-F1 from 0.80 to 0.82. By class, the model becomes more precise in W/Q and C, and is more willing to recover G and OTH. For C, precision increases from 0.82 to 0.88 while recall decreases from 0.82 to 0.79, but F1 is slightly higher (0.83). The model cuts false positives of C but misses a few more true C. G has a reverse pattern to C, with precision dropping from 0.89 to 0.85, but recall increases from 0.80 to 0.86, having an F1 of 0.85. While OTH shows a slight gain by recall from 0.62 to 0.66, with an F1 of 0.68, though it remains the hardest label. W/Q has a precision increment from 0.85 to 0.88 and a small recall drop from 0.93 to 0.92, and has an F1 of 0.90. In short, this tuning shows it trades a little aggressiveness for cleaner decisions, trims W/Q false positives, and improves recovery of Grounds and OTH, although OTH still possess the main gap in recall.

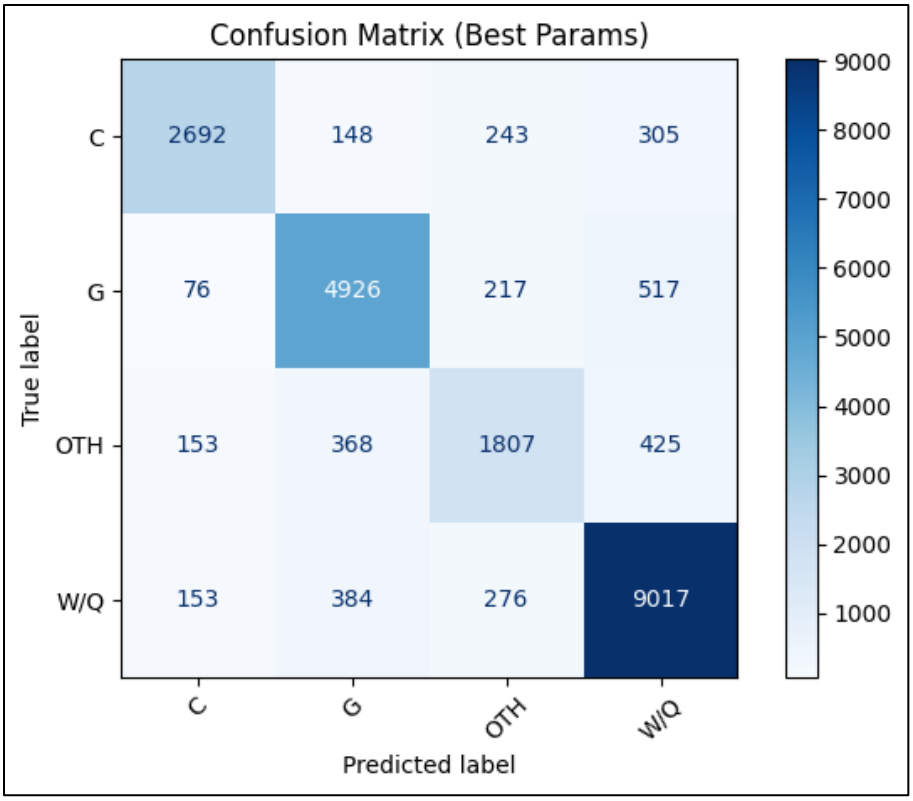


Figure 81: CNN Confusion Matrix (Best Tuned)

The first thing noted from the tuned CNN confusion matrix is a drop in spill into W/Q and C. Off-diagonal items landing in the W/Q column fall from 1,671 to 1,247, which matches the higher W/Q precision in the classification report. G benefit the most, as shown in the correct G increases from 4,563 to 4,926, while mistakes to W/Q fall from 787 to 517 and to OTH from 239 to 217. OTH also improves. True OTH increases from 1,714 to 1,807, and the flow to W/Q drops from 566 to 425, although more OTH shift to G, from 252 to 368. While C trades precision for recall. Fewer non-C items are sent to the C column. For example, G to C falls from 147 to 76 and W/Q to C from 230 to 153, so C predictions are cleaner, but true C decreases from 2,782 to 2,692, and more C moves to G and OTH. Correct W/Q drops from 9,168 to 9,017, and a few more W/Q move to G and OTH, but fewer false positives entering W/Q from other classes. Overall, tuning reduces the high classification to W/Q, increases the chance for G and OTH, and C precision.

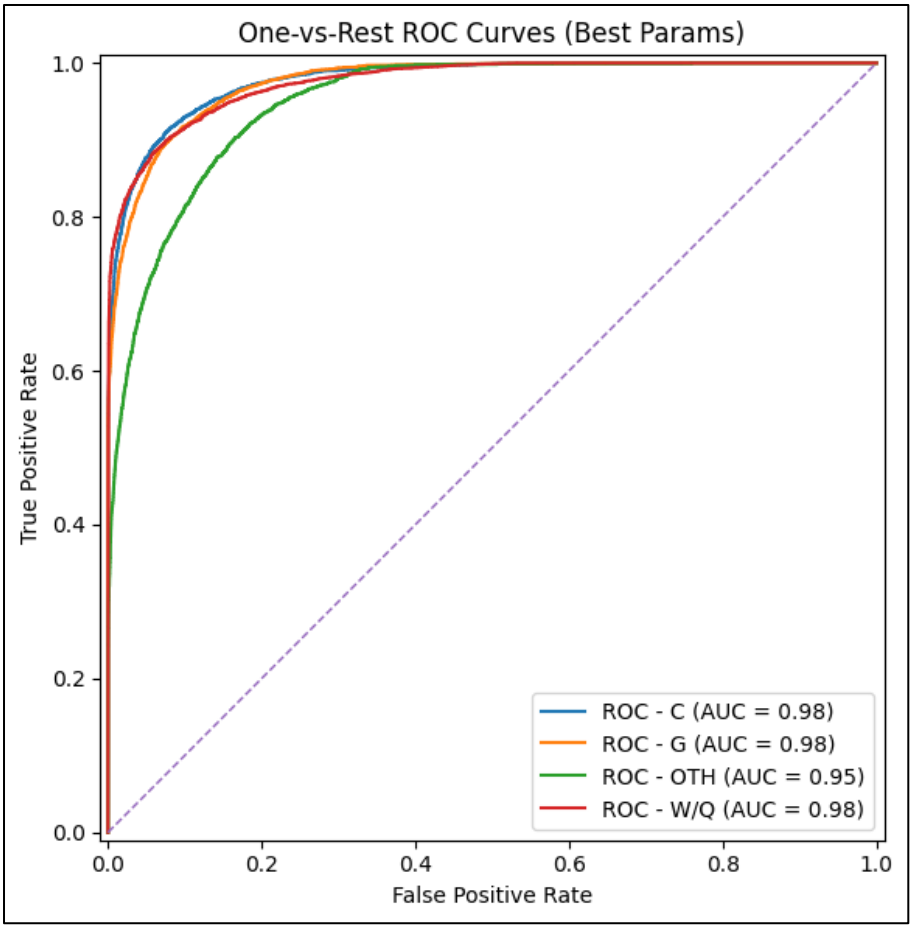


Figure 82: CNN ROC Chart (Best Tuned)

```
ROC-AUC (OvR, macro):    0.9695
ROC-AUC (OvR, weighted): 0.9725
```

Figure 83: CNN Macro and Weight ROC-AUC (Best Tuned)

The tuned ROC curves show the CNN separates classes well, and a little better than before. C, G and W/Q each reach an AUC near 0.98, which is a small increase for G and W/Q compared with the baseline (0.97). OTH stays the lowest at about 0.95, so it remains the hardest class, although its curve is still far from random. The macro-ROC-AUC moves from about 0.968 to 0.970, and the weighted ROC-AUC moves from about 0.971 to 0.973, which shows a small and modest gain in overall ranking ability. The curves attach close to the top left corner, meaning the model can hold a high true positive rate with few false errors for many different score cutoffs. Together with the confusion matrix, this indicates that the tuned model is already better at excluding non-WQ items from WQ.

Overall, the CNN is good at classifying argument types, and the tuned version makes it steadier and cleaner. Accuracy rises from 84% to 85% and macro-F1 from 0.80 to 0.82. W/Q becomes more precise with fewer false positives, Grounds recovers more true cases, and OTH is increased slightly in both recall and F1. Claims trade a small drop in recall for a clear increase in precision, as shown in the confusion matrix. The ROC view also improves slightly, with macro-AUC moving to about 0.970 and weighted AUC to about 0.973. The main gaps that remain are recall OTH.

5.3.3 DeBERTa

```
=== Classification Report ===
              precision    recall  f1-score   support

           C       0.90      0.94      0.92      3388
           G       0.94      0.94      0.94      5736
         W/Q       0.95      0.94      0.95      9830
         OTH       0.82      0.83      0.83      2753

    accuracy                           0.93     21707
   macro avg       0.90      0.91      0.91     21707
weighted avg       0.93      0.93      0.93     21707
```

Figure 84: DeBERTa Classification Report

DeBERTa delivers a very strong and well-balanced performance on the validation set. Overall accuracy is 93% with macro-F1 0.91. All classes perform well, not just the largest one. The result is amazing with the support of the training summary, which accuracy reached about 92.5% by the second epoch, and the validation scores improved at the same time, without a sign of overfitting. C are recovered very reliably with a recall of 0.94 and a precision of 0.90. While G is stable, which shows precision and recall both at 0.94. W/Q also achieves the best F1 at 0.95, pairing high precision 0.95 with a recall of 0.94. OTH is still the least certain label, but remains solid already at the precision of 0.82 and recall of 0.83 if the compared objects are the models evaluated previously. The weighted averages match the accuracy (93%). This shows consistency across the dataset size. In short, DeBERTa captures the structure of the Toulmin argument types well.

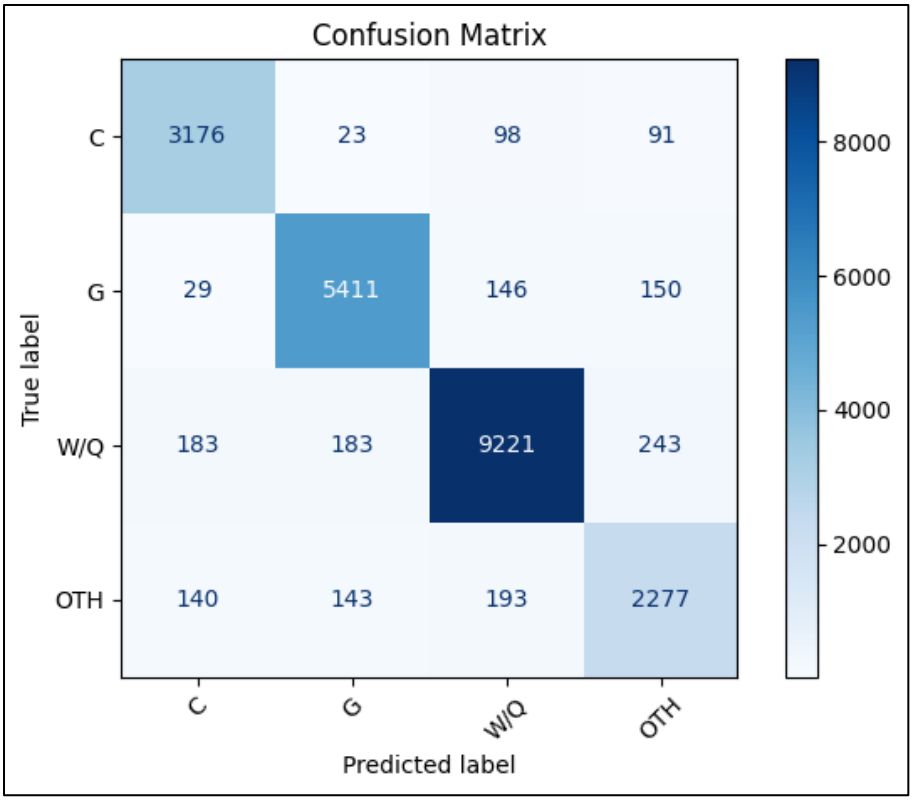


Figure 85: DeBERTa Confusion Matrix

The confusion matrix is highly diagonal, which means DeBERTa gets most sentences right for every class. C are very clean with 3,176 correct out of 3,388. G is also steady with 5,411 correct out of 5,736, and only a few classifications go to W/Q or OTH. W/Q is the largest row and has 9,221 correct out of 9,830. OTH is the least distinct among others, with 2,277 correct out of 2,753; the main mistakes move to WQ and to a lesser extent to G and C. The matrix columns explain the high precision as shown in the classification report. When the model predicts C, about 9 out of 10 are truly C. Predictions of G are right about 94% of the time, and the remaining classes too, with high precision. Overall, DeBERTa keeps errors small.

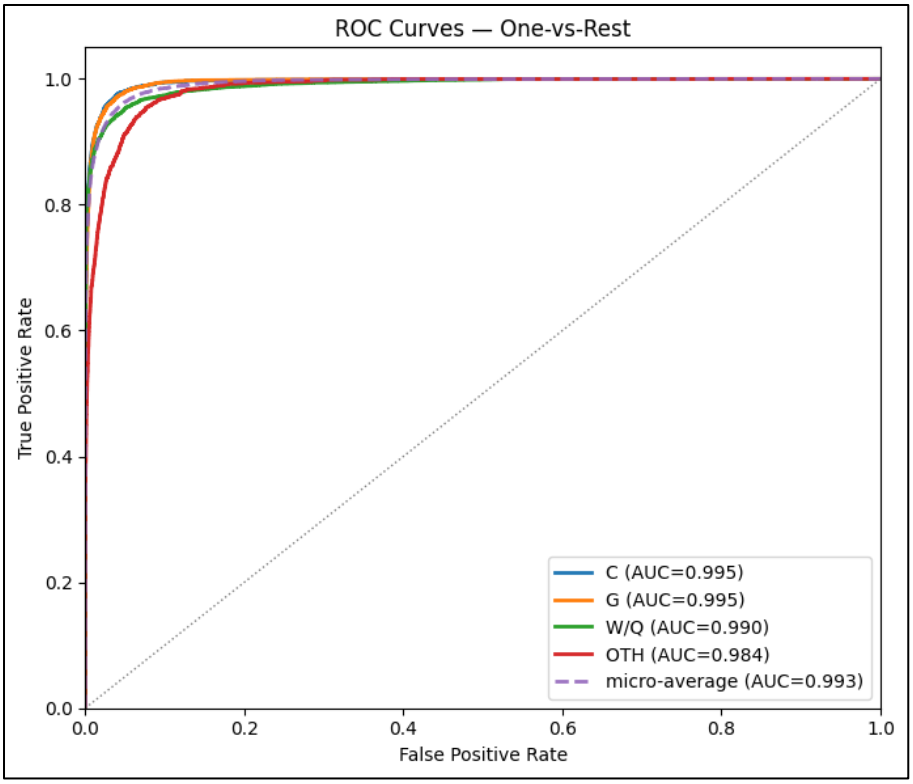


Figure 86: DeBERTa ROC Chart

```
=== ROC AUCs ===
C: AUC = 0.9949
G: AUC = 0.9949
W/Q: AUC = 0.9902
OTH: AUC = 0.9841
Micro-average AUC: 0.9927
Macro-average AUC: 0.9910
```

Figure 87: DeBERTa ROC-AUC Summary

The ROC curves show that the DeBERTa model is good at separating each class from the others across all score cutoffs. The areas under the curves are very high for every class, with C and G at about 0.995, W/Q at about 0.990, and OTH at about 0.984. Values this close to one indicate that the model can maintain high true positives while keeping false alarms low across a wide range of thresholds. The micro-average AUC is about 0.993, which reflects overall ranking across all sentences, and the macro-average is about 0.991, which treats each class equally and confirms that performance is strong.

```
=== Classification Report (Full Val) ===
              precision    recall  f1-score   support

           C       0.89      0.91      0.90      3388
           G       0.93      0.93      0.93      5736
         W/Q       0.94      0.93      0.94      9830
         OTH       0.79      0.80      0.79      2753

    accuracy                           0.91     21707
   macro avg       0.89      0.89      0.89     21707
weighted avg       0.91      0.91      0.91     21707
```

Figure 88: DeBERTa Classification Report (Best Tuned)

The best-tuned DeBERTa is also strong, but the performance drops a little compared with the baseline model. Overall accuracy is 91% and macro-F1 is 0.89. C, G and W/Q stay high at 0.90, 0.93 and 0.94 F1-score, while OTH drops the most to 0.79 F1-score. This pattern suggests that the tuned settings made the model slightly less able to pick up the less distinctive phrases in OTH cases. The possible reason is that hyperparameters are selected through tuning with a small sample of about 100 debates due to time and resource limitations, so the search may not have captured the full variety in the validation set, which provides powerful settings for the full retrain. Small differences such as sequence length, warmup, and weight decay chosen on the subset can lead to a little change in performance.

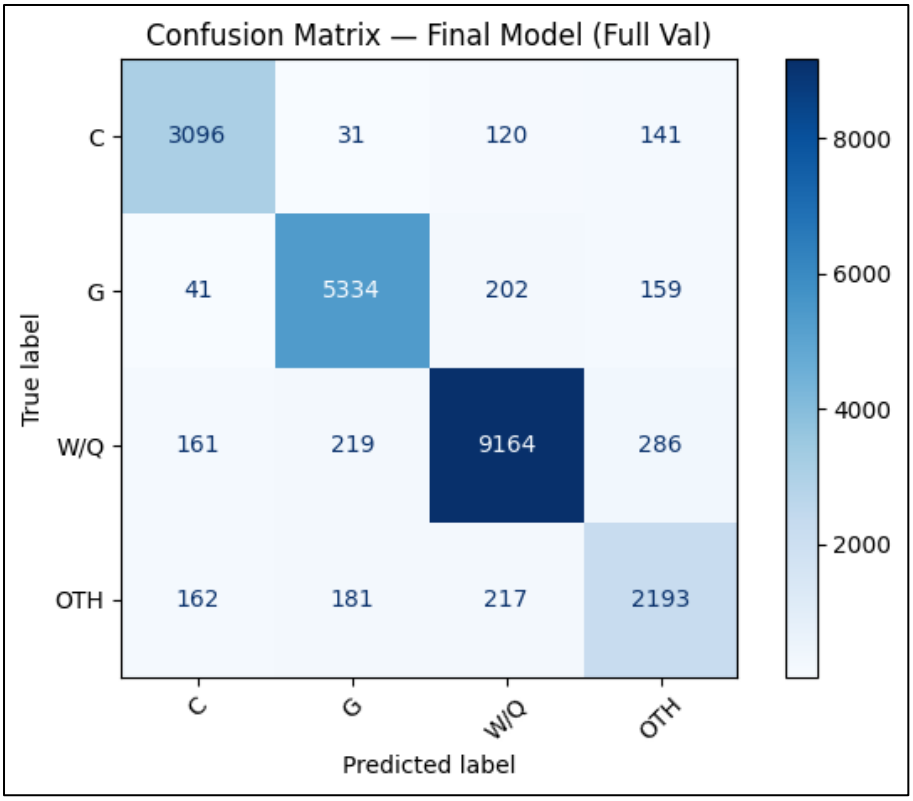


Figure 89: DeBERTa Confusion Matrix (Best Tuned)

The confusion matrix shows that most sentences are classified correctly. C have 3,096 correct out of 3,388. G is similarly solid at 5,334 of 5,736. W/Q remains the most reliable row with 9,164 of 9,830 correct. OTH, on the other hand, reached 2,193 true labels out of 2,753. The classified C totals 3,460 items, and about 90% are truly C, while classified G totals 5,765 with about 93% true, total classified W/Q is 9,703 with about 94% true, and lastly, OTH totally classifies 2,779 with about 79% true. The main weakness is that the two-way confusion between OTH and the argumentative roles, especially W/Q and G, aligns with the lower OTH F1-score in the report and explains most of the small drop after tuning.

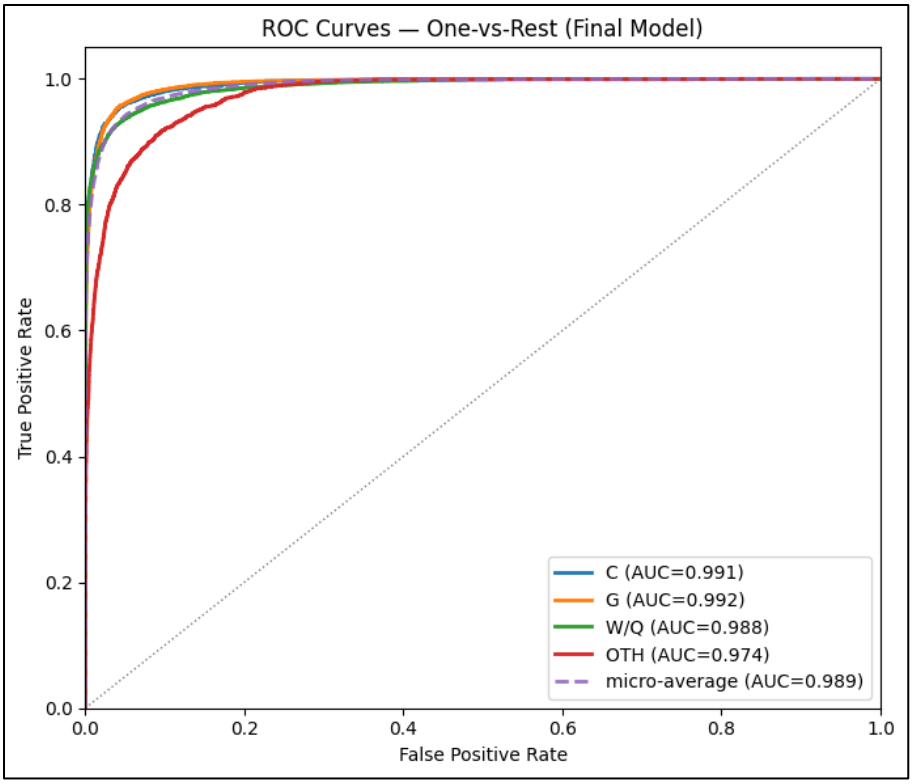


Figure 90: DeBERTa ROC Chart (Best Tuned)

```
=== ROC AUCs (Full Val) ===
C: AUC = 0.9913
G: AUC = 0.9918
W/Q: AUC = 0.9876
OTH: AUC = 0.9742
Micro-average AUC: 0.9888
Macro-average AUC: 0.9862
```

Figure 91: DeBERTa ROC-AUC Summary (Best Tuned)

The ROC curves show that the tuned DeBERTa also separates all classes well across score cutoffs. AUCs are very high for every class, with C at about 0.991, G at about 0.992, W/Q at about 0.988, and OTH at about 0.974. The micro-average AUC is about 0.989, and the macro-average is about 0.986. Overall ranking remains good even when each class is weighted equally. Compared with the earlier baseline DeBERTa run, these AUCs drop only slightly.

At this point, DeBERTa has stood out as the most reliable Toulmin argument type classifier. The baseline run yields high and balanced scores (accuracy of approximately 93% and macro-F1 of about 0.91) with a mostly diagonal confusion matrix and AUCs of nearly 0.99 across classes. After

getting hyperparameters from tuning a small subset and retraining, performance drops slightly (accuracy about 91% and macro-F1 about 0.89). The drop possibly reflects hyperparameters chosen on too small a sample rather than a limitation of the model. Overall, as the first transformer model trained, DeBERTa achieves significantly better performance in capturing debate sentence structure compared to the previous study.

5.3.4 GPT

```
=== Classification Report ===
              precision    recall  f1-score   support

           C       0.84      0.92      0.88      3388
           G       0.90      0.91      0.91      5736
         W/Q       0.94      0.91      0.92      9830
         OTH       0.77      0.74      0.75      2753

    accuracy                           0.89     21707
   macro avg       0.86      0.87      0.87     21707
weighted avg       0.89      0.89      0.89     21707
```

Figure 92: GPT Classification Report

The GPT argument type classifier shows balanced behaviour with 89% accuracy and macro-F1 of 0.87. The performance is good across all labels. C are recovered very well with a recall of 0.92, though the precision of 0.84 suggests some non-claims are pulled into C, likely when sentences contain assertive phrasing without clear claim markers. G is clean at 0.90 precision and 0.91 recall. W/Q is the best class overall with 0.94 precision and 0.91 recall. OTH remains the weakest at 0.77 precision and 0.74 recall. The weighted averages match the headline accuracy of 0.89, which confirms consistency across the dataset size. In short, GPT captures the structure of G, G, and W/Q well, and finds OTH the most challenging, as is the case in other models.

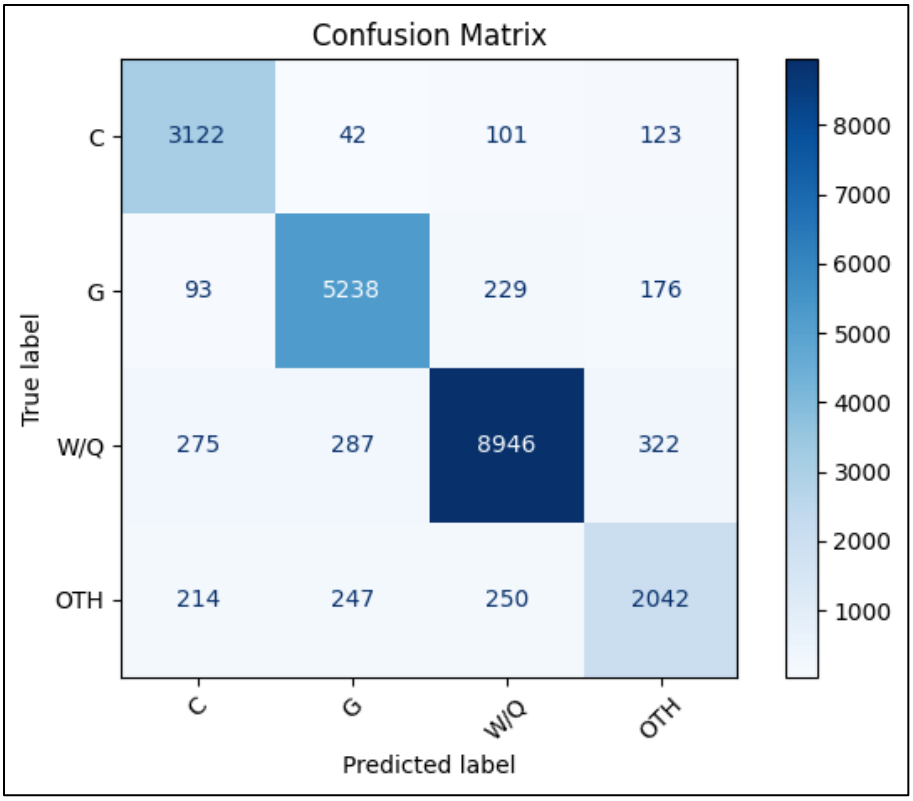


Figure 93: GPT Confusion Matrix

With an overall accuracy of 89%, the confusion matrix is mostly diagonal. For C, 3,122 of 3,388 are right (92% recall). The rest drift mainly to W/Q with 101 cases (3%) and to OTH with 123 cases (3.6%), and others going to G. G also has 5,238 out of 5,736, almost 91%, with most misses sliding to W/Q at 229 cases and to OTH at 176 cases. While W/Q has 8,946 of 9,830 correct, 9% of them are mistaken as other classes. Its errors are spread thinly across C, G, and OTH at roughly 3% each. OTH is the toughest row with 2,042 of 2,753 correct, and the largest leaks are to W/Q at 9.1% and to G at 9%, with 7.8% moving to C. While looking at the precision by column, the classification of C is 3,704 in total, with 84% truly C, G has 5,814 of truly G, W/Q has 9,526 of truly W/Q, and OTH has 2,663 of truly OTH. GPT is also strong on C, G, and W/Q, while OTH is the source of confusion.

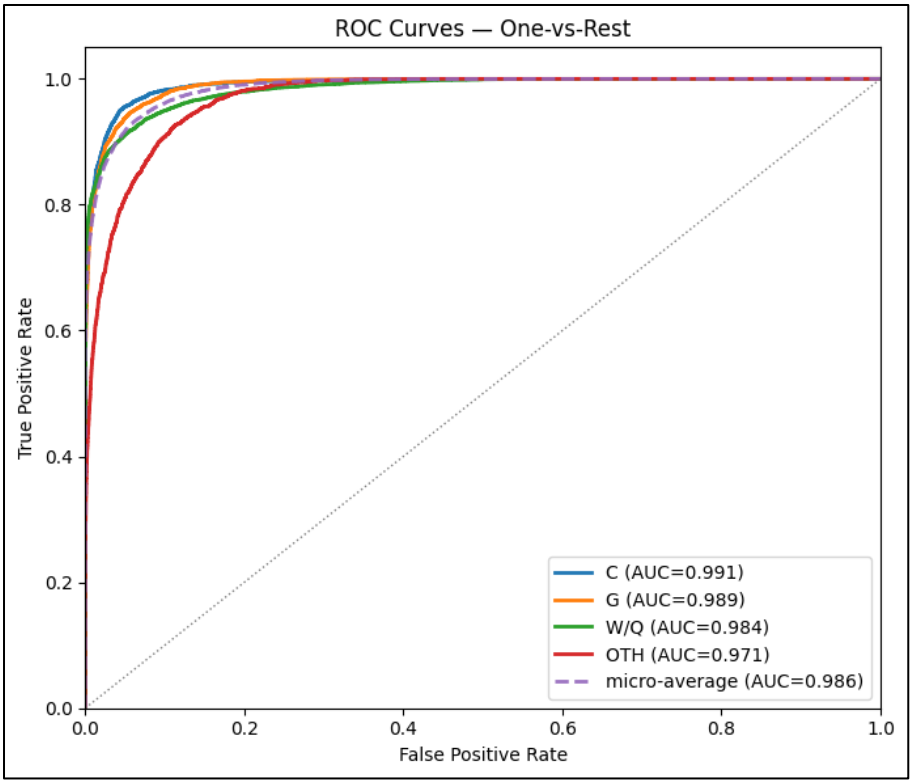


Figure 94: GPT ROC Chart

```
=== ROC AUCs ===
C: AUC = 0.9906
G: AUC = 0.9893
W/Q: AUC = 0.9838
OTH: AUC = 0.9706
Micro-average AUC: 0.9863
Macro-average AUC: 0.9836
```

Figure 95: GPT ROC-AUC Summary

The ROC curves show that C and G have the strongest areas under the curve at about 0.991 and 0.989. W/Q follows closely at about 0.984, and OTH is the weakest among them at about 0.971, yet this is still a strong result. The micro-average AUC is about 0.986, and the macro-average AUC is about 0.984, which weights each class equally and drops slightly because of OTH. These curves are all near the top left corner. In short, the GPT baseline model offers good class separability. Most of the remaining improvement is likely to come from class-specific thresholds rather than large architectural changes.

```
=== TUNED CLASSIFICATION REPORT ===
              precision    recall  f1-score   support

           C       0.85      0.85      0.85      3388
           G       0.85      0.88      0.86      5736
         W/Q       0.91      0.91      0.91      9830
         OTH       0.72      0.67      0.69      2753

    accuracy                           0.86     21707
   macro avg       0.83      0.83      0.83     21707
weighted avg       0.86      0.86      0.86     21707
```

Figure 96: GPT Classification Report (Best Tuned)

The best-tuned GPT model steps down from the baseline. Overall accuracy moves from 89% to 86% and macro-F1 from 0.87 to 0.83. By class, the biggest shifts are C and OTH. C keeps similar precision (from 0.84 to 0.85), but recall falls from very high, from 0.92 to 0.85, which makes F1 drop from 0.88 to 0.85. The model becomes less willing to call borderline sentences a C. G with precision 0.85, recall 0.88, and 0.86 F1 also becomes weaker, compared to the previous result of 0.9 precision, 0.91 recall, and 0.91 F1-score. W/Q is stable at 0.91 precision, recall, and F1, which means the core reasoning cue patterns are still captured. OTH decreases the most from 0.75 to 0.69 F1, driven by both lower precision (from 0.77 to 0.72) and lower recall (from 0.74 to 0.67). It has already been examined through DeBERTa evaluation and now confirmed that tuning on a small subset likely chose settings that do not generalise to the full validation mix.

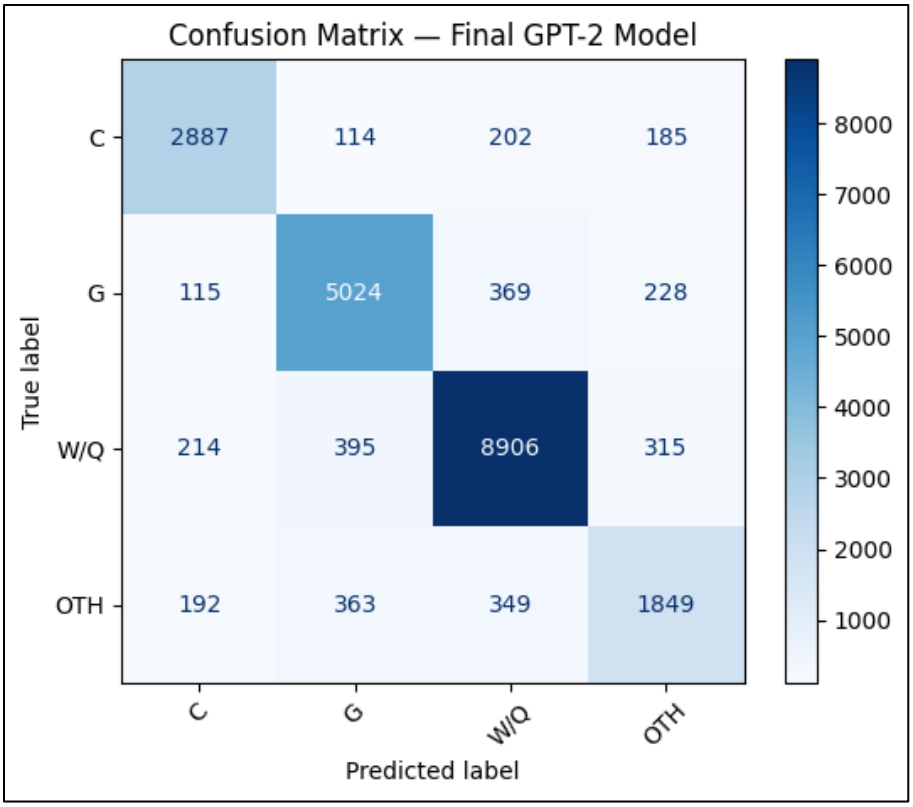


Figure 97: GPT Confusion Matrix (Best Tuned)

The tuned GPT matrix shows that most of the correct hits are still on the diagonal, but the recall for C and OTH slips. C drops from 3,122 to 2,887 correct, so recall moves from about 92% to about 85%, with more C sentences now drifting to W/Q and OTH (about 6% and 5.5% of all C, up from 3% and 3.6%). G is also falling from 5,238 to 5,024, correct. Its largest error path into W/Q grows to about 6.4% of all G. The correct classification of W/Q drops a little from 8,946 to 8,906. OTH is where the biggest change appears. Correct OTH falls from 2,042 to 1,849, and misclassifications into G and W/Q jump to about 13.2% and 12.7% of all OTH. When analysed by the column for precision, the performance also drops, with all classes having decreased in total correct classification cases. In short, the best hyperparameters taken from 100 debate transcript training for retraining the GPT do not seem to work well.

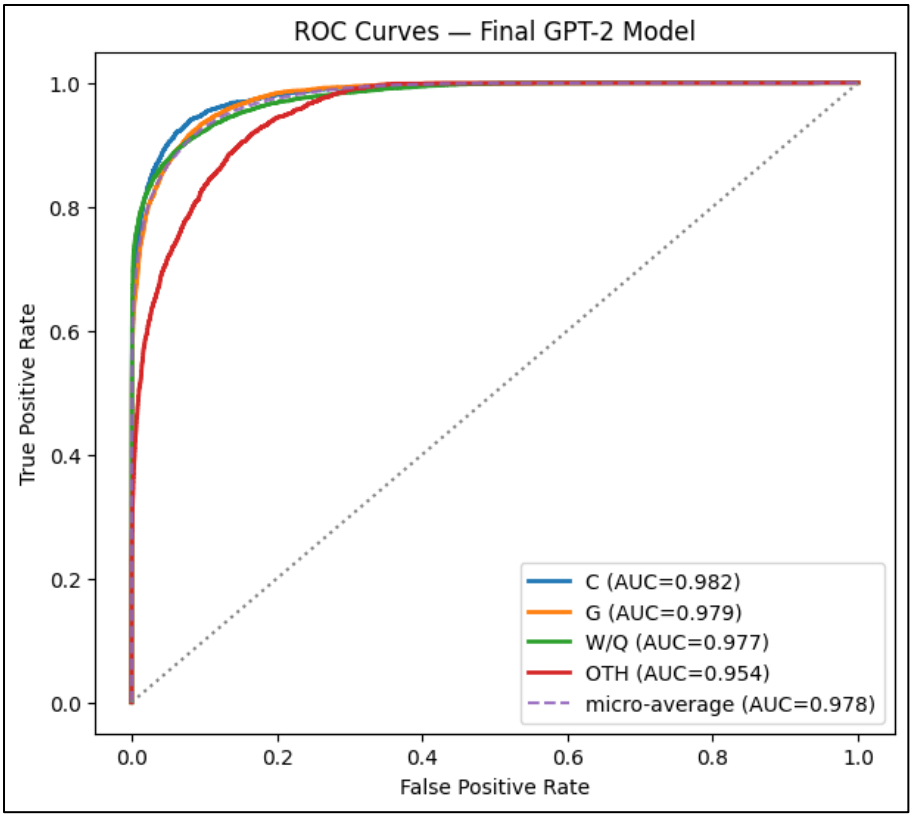


Figure 98: GPT ROC Chart (Best Tuned)

```
=== ROC AUCs (Tuned Val) ===
C: AUC = 0.9820
G: AUC = 0.9790
W/Q: AUC = 0.9771
OTH: AUC = 0.9538
Micro-average AUC: 0.9776
Macro-average AUC: 0.9730
```

Figure 99: GPT ROC-AUC Summary (Best Tuned)

The best hyperparameter-tuned GPT ROC curves are lower than the baseline. C drops from about 0.991 AUC to 0.982, G from about 0.989 to 0.979, and W/Q from about 0.984 to 0.977. The largest change is OTH, moving from about 0.971 to 0.954. The micro-average falls from about 0.986 to 0.978, and the macro-average from about 0.984 to 0.973. The curves are still okay above the diagonal. The ranking remains strong, but the headroom is smaller than before tuning. Since AUC is threshold-free, this decline indicates genuine changes in separability. Likely, the hyperparameters chosen on the small tuning subset did not generalise well.

Overall, the second transformer training, GPT, is also a capable classifier, but its best balance shows up in the baseline run. The baseline has high accuracy and macro-F1 with strong control of C, G, and W/Q, while OTH remains the hardest label. After tuning, the accuracy and macro-F1 drop; the confusion matrix also shows how the precision and recall are affected.

5.3.5 DeepSeek

```
=== Classification Report ===
              precision    recall  f1-score   support

           C       0.88      0.88      0.88      3388
           G       0.89      0.89      0.89      5736
         W/Q       0.91      0.92      0.92      9830
         OTH       0.75      0.73      0.74      2753

    accuracy                           0.88     21707
   macro avg       0.86      0.85      0.86     21707
weighted avg       0.88      0.88      0.88     21707
```

Figure 100: DeepSeek Classification Report

The DeepSeek baseline performs with 88% accuracy and a macro-F1 of 0.86. The results are balanced for C and G, each at 0.88 and 0.89 F1-score, which shows the model both finds most true cases (recall about 0.88 and 0.89) and keeps false alarms under control (precision about 0.88 and 0.89). W/Q is the strongest among other classes at 0.92 F1 with a recall of 0.92. The weak spot is OTH at 0.74 F1 with precision 0.75 and recall 0.73, which is expected because OTH is a mixed, less distinctive category and often gets pulled toward the argumentative roles, similar to other models' results. The close match between 88% accuracy and 0.88 weighted F1 suggests stable behaviour across the dataset size.

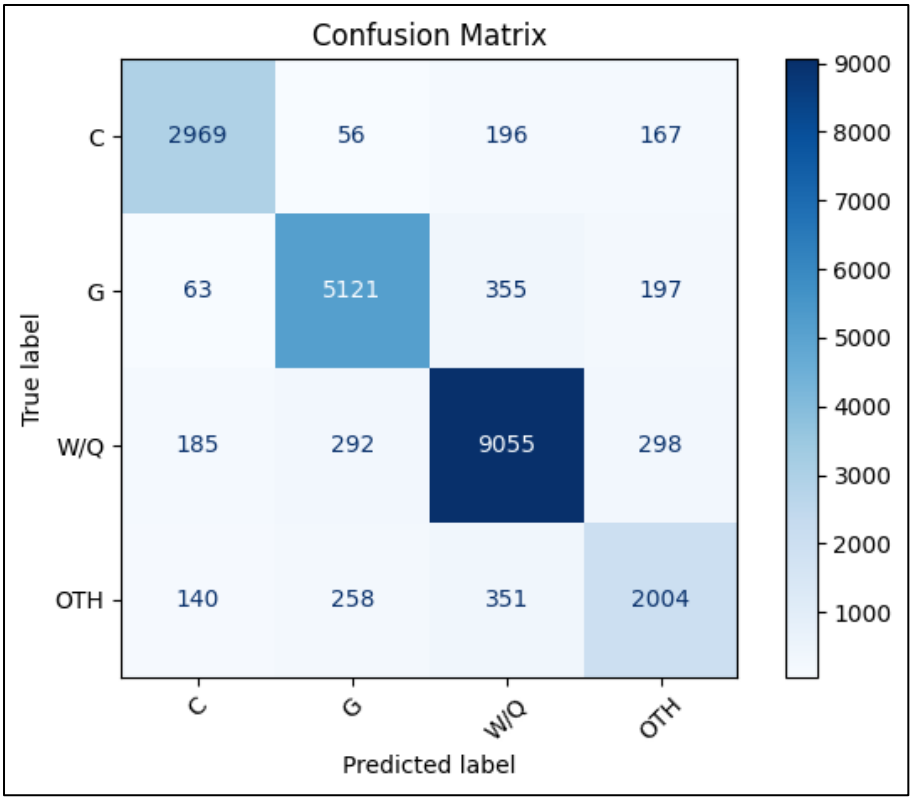


Figure 101: DeepSeek Confusion Matrix

The confusion matrix indicates that DeepSeek achieves a high accuracy rate, with a total of 19,149 correct classifications, contributing to an overall accuracy of approximately 88%. The diagonal cells show the correct hits, including C with 2,969, G with 5,121, W/Q with 9,055, and OTH with 2,004 correct. For true C, recall is about 88%. Around 5.8% of all C are mistaken as W/Q, with smaller proportions going to OTH (4.9%) and G (1.7%). The correctly classified G has 5,121 cases, but 355 cases, about 6.2%, still drift into W/Q, and 197 (3.4%) into OTH. W/Q is the most reliable row, with a recall rate of about 92%. Its misses are small, with 185 cases assigned to C (1.9%), 292 to G (3.0%), and 298 to OTH (3.0%). Classified OTH only has about 73% are true, while 351 move to W/Q and 258 move to G. That volume supports why W/Q recall is very high, while its precision sits around 91%.

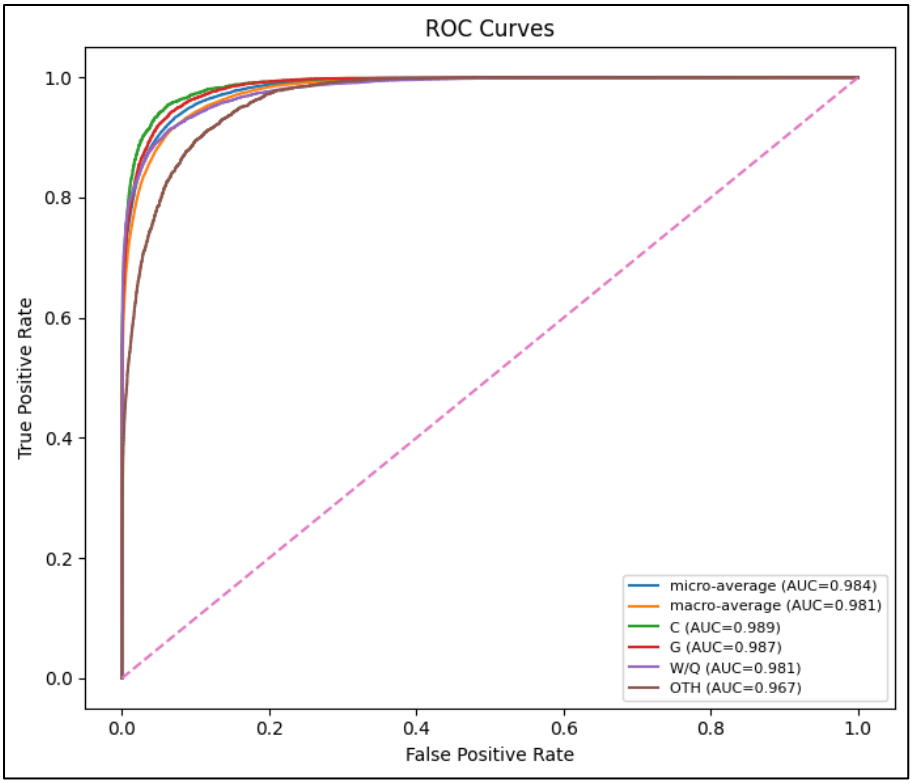


Figure 102: DeepSeek ROC Chart

The ROC curves indicate how DeepSeek separates the four labels across many score cutoffs. All four classes have the curves that almost attach to the top-left corner, which means the model can keep a high true-positive rate while holding down false positives. The AUCs are high for every class, including C with 0.989, G with 0.987, W/Q with 0.981, and OTH with 0.967. The micro-average is about 0.984, and the macro-average is about 0.981. This indicates a strong overall ranking, with only a slight drop when each class is weighted equally. The lower AUC for OTH matches its role as the most ambiguous category, but 0.967 is already good enough. In short, the ROC chart shows a good classifier with separability and clear, targeted options to rebalance errors by class.

```
=== FINAL CLASSIFICATION REPORT ===
              precision    recall  f1-score   support

           C       0.90      0.89      0.90      3388
           G       0.89      0.93      0.91      5736
         W/Q       0.94      0.92      0.93      9830
         OTH       0.76      0.75      0.76      2753

    accuracy                           0.90     21707
   macro avg       0.87      0.87      0.87     21707
weighted avg       0.90      0.90      0.90     21707
```

Figure 103: DeepSeek Classification Report (Best Tuned)

The DeepSeek model tuned with the optimal hyperparameters exhibits a notable performance improvement. Overall accuracy rises from 88% to 90% and macro-F1 from 0.86 to 0.87. C improves from 0.88 to 0.90 F1 with 0.90 precision and 0.89 recall. G gains the most with recall jumps from 0.89 to 0.93 and F1 to 0.91. W/Q increases slightly, moving to 0.93 F1 with higher precision of 0.94 and recall of 0.92. OTH, which is the hardest label, also increases from F1 of 0.74 to 0.76, with both precision and recall slightly higher (0.76 and 0.75). Weighted averages match the headline accuracy (0.90), so the gains are not confined to one class. In short, tuning makes DeepSeek more balanced, which recovers more G and OTH, and improves precision for W/Q and C.

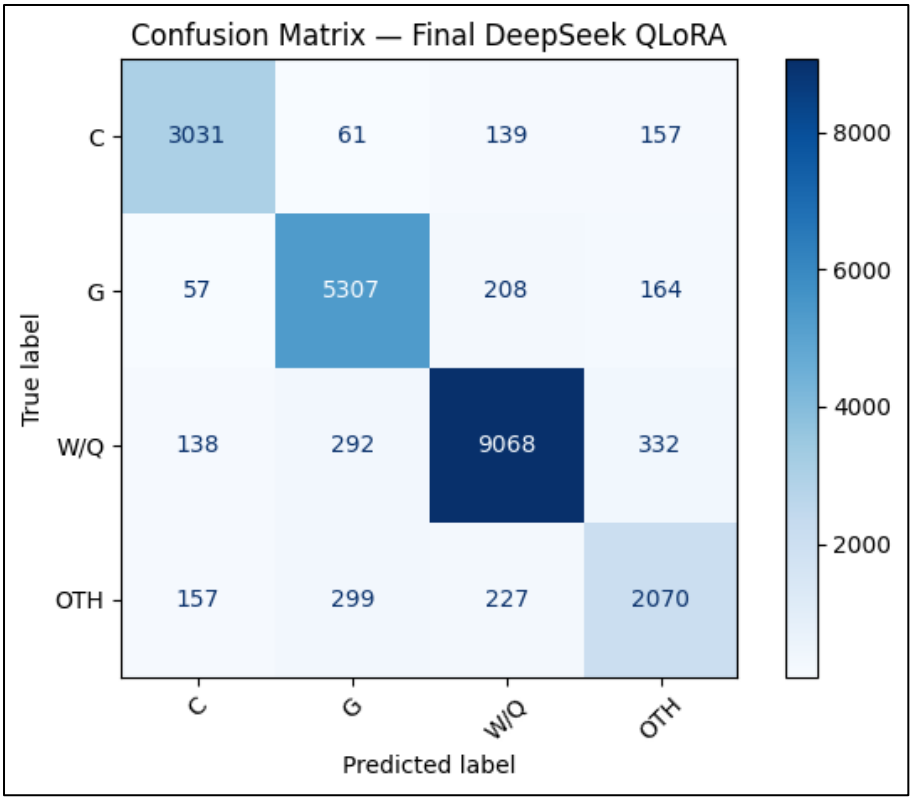


Figure 104: DeepSeek Confusion Matrix (Best Tuned)

The tuned DeepSeek with QLoRA model, trained with the best hyperparameters, exhibits a tighter diagonal and fewer classification errors than the baseline, indicating clearer class boundaries. True positives increase across all classes, including C from 2,969 to 3,031, G from 5,121 to 5,307, W/Q from 9,055 to 9,068, and OTH from 2,004 to 2,070. The confusions are also reduced obviously. For example, true G mistakenly classified to W/Q dropped from 355 to 208, true OTH to W/Q from 351 to 227, and true C to W/Q from 196 to 139. This reduces the prior tendency to over-classify W/Q and increases precision for that class, as well as improving recall for C and G. However, note that true W/Q misclassify to OTH rises from 298 to 332 and true OTH to G from 258 to 299. This shows that OTH is still blurred with weak evidence. Overall, the confusion matrix shows an improvement in correctness and a more balanced distribution of misclassifications.

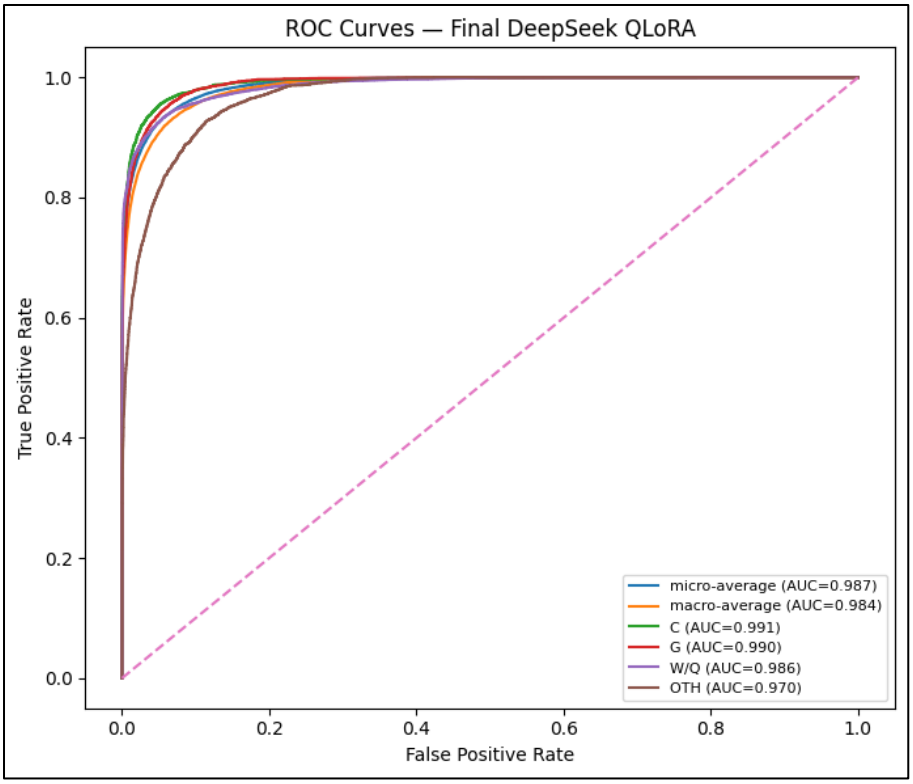


Figure 105: DeepSeek ROC Chart (Best Tuned)

The ROC curves for the best hyperparameters tuned DeepSeek with QLoRA model increase a bit and are left relative to the baseline, which indicates better separability at low false-positive rates. AUC values improve consistently, with micro-average increases from 0.984 to 0.987 and macro-average from 0.981 to 0.984. While per-class AUC increases for C (0.989 to 0.991), G (0.987 to 0.990), W/Q (0.981 to 0.986), and OTH (0.967 to 0.970). The largest improvement is W/Q, which aligns with the confusion-matrix reduction in over-classifying W/Q. OTH remains the weakest curve despite a small improvement. In short, compared to the baseline model, the tuned model not only lifts overall ranking quality (micro) but also improves performance across classes (macro).

After tuning on 100 transcripts and retraining on the full corpus, DeepSeek QLoRA shows improvement over the baseline. Accuracy rises from 88% to 90%, macro-F1 from 0.86 to 0.87, and weighted F1 from 0.88 to 0.90. F1-score also improves for every class. Confusion patterns show fewer mislabels into W/Q, especially for G and C, and ROC AUC increases for micro from 0.984 to 0.987 and macro from 0.981 to 0.984. Overall, the tuned model is more accurate and balanced.

5.3.6 LLaMA

```
=== Classification Report (LLaMA QLoRA) ===
              precision    recall  f1-score   support

           C       0.89      0.88      0.88      3388
           G       0.90      0.91      0.91      5736
         W/Q       0.91      0.93      0.92      9830
         OTH       0.78      0.72      0.75      2753

    accuracy                           0.89     21707
   macro avg       0.87      0.86      0.86     21707
weighted avg       0.89      0.89      0.89     21707
```

Figure 106: TinyLLaMA Classification Report

The classification report of baseline model TinyLLaMA with QLoRA is balanced with 89% accuracy and macro-F1 of 0.86. The performance is strong across all classes. C are steady at a precision of 0.89 and a recall of 0.88. G is also reliable with 0.90 precision, 0.91 recall, and 0.91 F1. W/Q is the strongest class, pairing high precision 0.91 with very high recall 0.93 for 0.92 F1. The weak point is OTH, which has 0.78 precision, 0.72 recall and 0.75 F1. The weighted averages align with the headline accuracy, which suggests stable behaviour across the dataset size. The model looks dependable out of the box because the most direct gains would come from recovering more OTH.

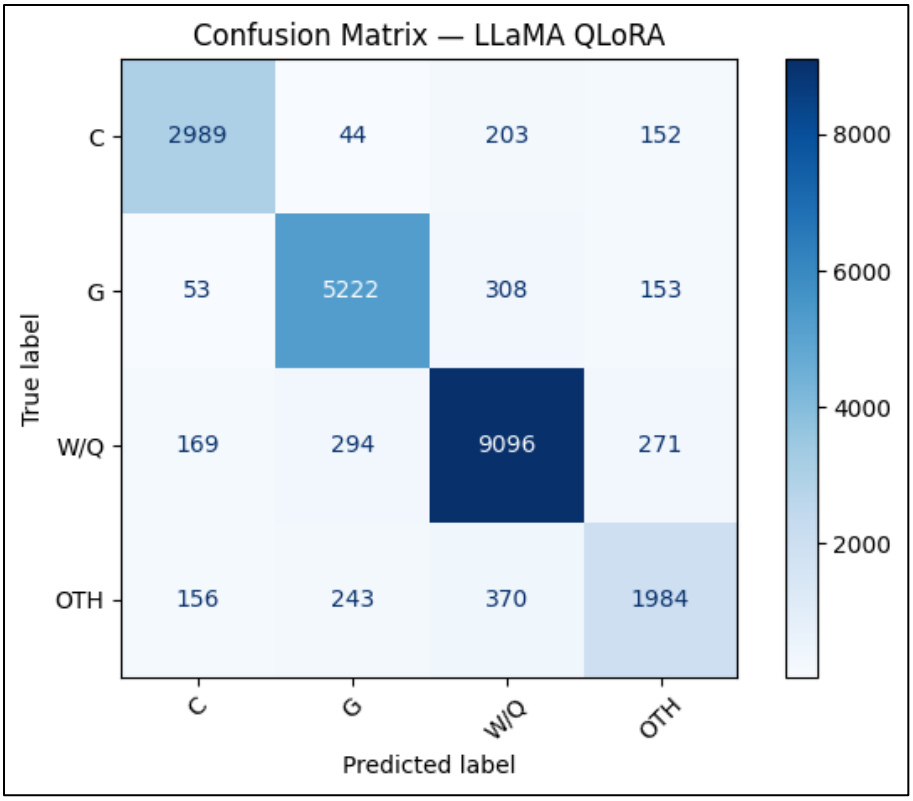


Figure 107: TinyLLaMA Confusion Matrix

The TinyLLaMA with QLoRA baseline model shows a strong diagonal with high true positives for W/Q, with 9,096 (93% recall) correct out of 9,830, G with 5,222 (91%) out of 5,736, and C with 2,989 (88%) out of 3,388, while OTH has 1,984 correct out of 2,753, which is only about 72% precision. Many sentences from C (203), G (308), and OTH (370) are pulled into W/Q, which suggests that the model tends to interpret varied argumentative content as W/Q. Confusion between C and G is comparatively small (44 in true C to G, 53 in true G to C). Both categories are reasonably well separated. OTH is classified into all three argumentative classes with 156 C, 243 G, and 370 W/Q. In short, the performance of the baseline model is okay on core argumentative roles, especially W/Q and G, but struggles in slightly over-classifying to W/Q, and does not classify true OTH well.

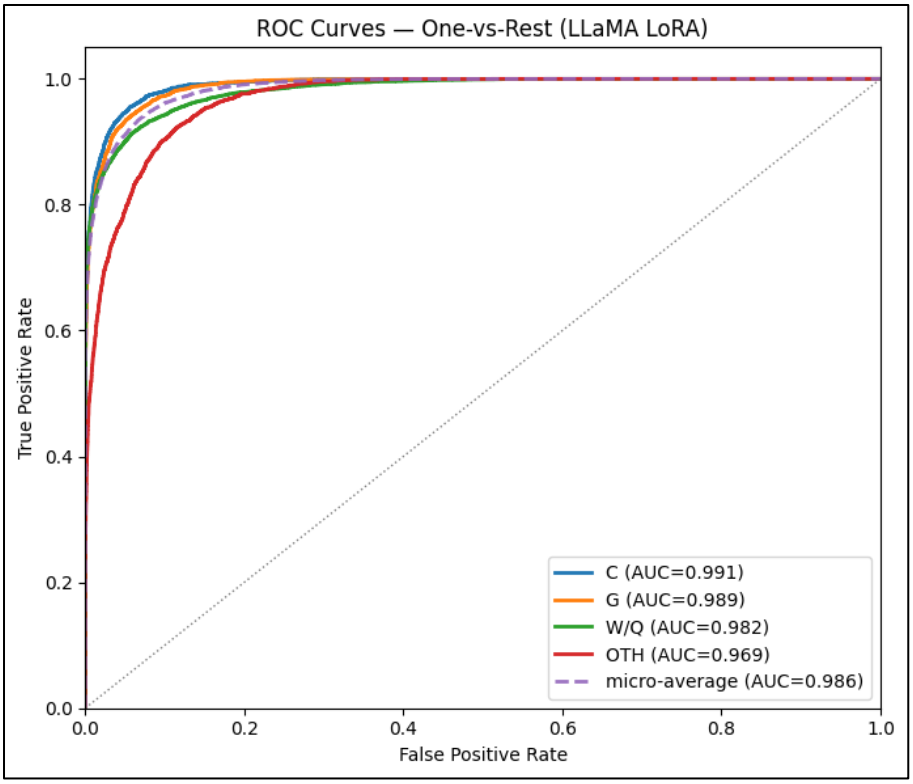


Figure 108: TinyLLaMA ROC Chart

The ROC curves show that TinyLLaMA separates the four labels well across a wide range of score cutoffs. Firstly, the lines stay close to the top left corner, indicating the model keeps true positives high while keeping false alarms low. C and G have the strongest areas under the curve, about 0.991 and 0.989. They are easy for the model to rank above the other classes. W/Q is also strong with an AUC of 0.982, and OTH at about 0.969. The micro-average AUC is about 0.986. It summarises a strong overall ranking across all sentences.

```
=== FINAL REPORT (TinyLLaMA QLoRA) ===
              precision    recall  f1-score   support

           C       0.92      0.90      0.91      3388
           G       0.92      0.93      0.93      5736
         W/Q       0.95      0.94      0.94      9830
         OTH       0.79      0.80      0.80      2753

    accuracy                           0.92     21707
   macro avg       0.89      0.89      0.89     21707
weighted avg       0.92      0.92      0.92     21707
```

Figure 109: TinyLLaMA Classification Report (Best Tuned)

The retrained TinyLLaMA with the best-tuned settings yields a classification report, with overall accuracy increasing from 89% to 92% and macro-F1 score rising from 0.86 to 0.89. Every class improves. C moves from a precision of 0.89 to 0.92, and a recall of 0.88 to 0.90, giving a 0.91 F1-score. G increases the F1-scores from 0.91 to 0.93 with both precision and recall at about 0.92 and 0.93 (0.01 increment each). W/Q improves the most on precision, increasing from 0.91 to 0.95, while recall also increases to 0.94. The most significant recovery is OTH, where recall rises from 0.72 to 0.80 and F1 reaches 0.80. This model has reduced the earlier drift of OTH into the other argumentative classes. In short, the overall model performance increases after retraining with the best hyperparameters.

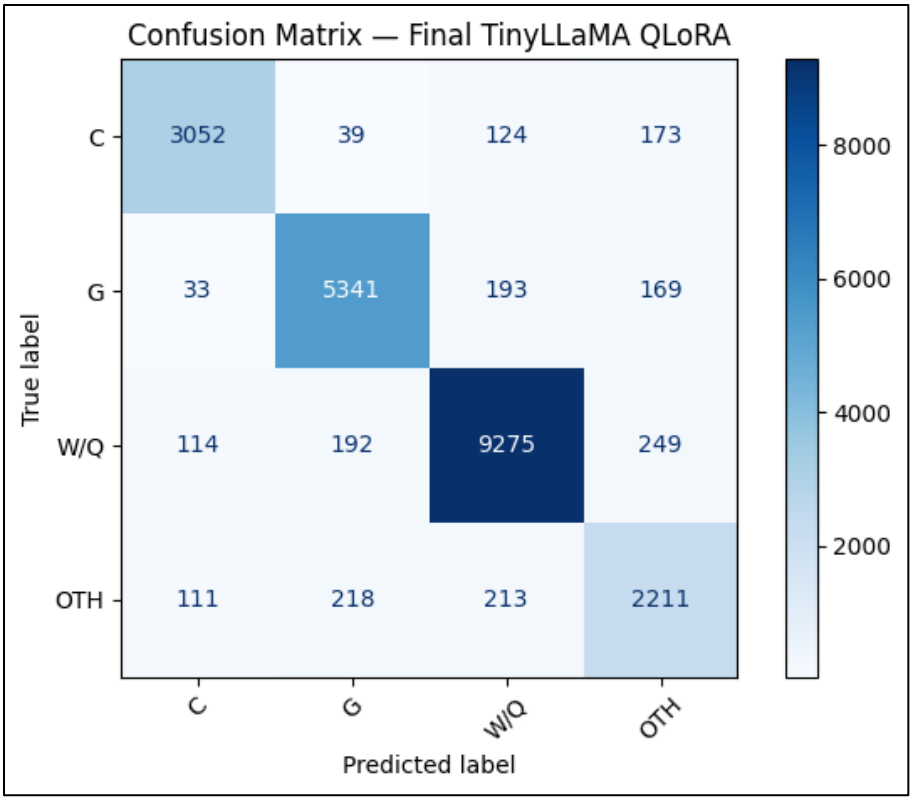


Figure 110: TinyLLaMA Confusion Matrix (Best Tuned)

The retrained TinyLLaMA with the best hyperparameters' confusion matrix is cleaner than the baseline. Correct hits rise in every row. C goes from 2,989 to 3,052, G from 5,222 to 5,341, W/Q from 9,096 to 9,275, and OTH from 1,984 to 2,211. The problem when many classes mistaken as W/Q become better. Off-diagonal items go into the W/Q column, dropping from 881 to 530, which is a reduction of about 40%. While C to W/Q falls from 203 to 124, G is classified as W/Q from 308 to 193, and OTH into W/Q from 370 to 213. The effect improves the recall accordingly. Precision also tightens when looking down the columns, the same as shown in the classification report. There is a small reverse for C with a few more C to OTH errors (from 152 to 173), but the proportion is very small. Overall, the tuned model shows stronger separation and better balance across all four classes.

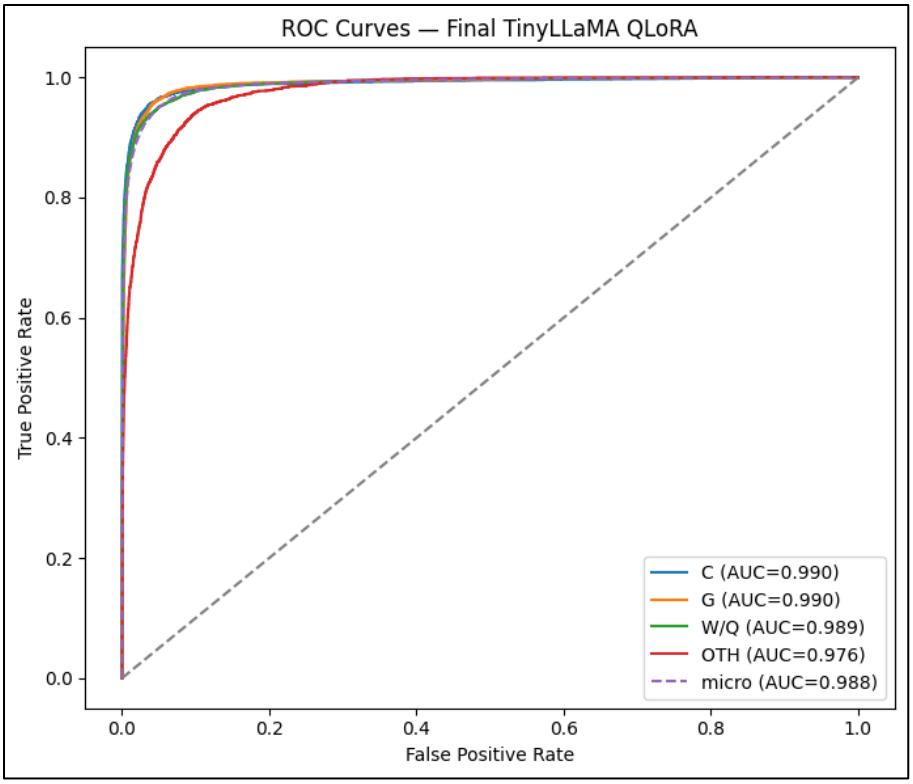


Figure 111: TinyLLaMA ROC Chart (Best Tuned)

The tuned TinyLLaMA ROC curves are close to the top left. This shows good separation across score cutoffs. The AUCs are high for every class and improve. WQ increases a little bit from about 0.982 to 0.989. OTH also rises from about 0.969 to 0.976. G increases to about 0.990, and C is unchanged at about 0.990. The micro-average increases from roughly 0.986 to 0.988, which is a small but real overall gain. The curves and AUCs align with the improvements in the tuned confusion matrix, which explain the stronger balanced performance.

To conclude, TinyLLaMA demonstrates strong and balanced performance through its baseline training, and after tuning, it becomes a clearly better all-rounder. Accuracy and macro-F1 score rise to about 92% and 0.89, respectively. The confusion matrix shows that fewer sentences are being classified into the wrong class. C and G also tighten up, with cleaner decisions and higher recalls. The ROC curves especially highlight the gains for W/Q and OTH, which become closer to the top-left corner after tuning. In short, tuning strengthens separation for every class and delivers a well-balanced model for argument type classification.

5.4 Model Comparison

After evaluating and discussing each baseline and tuned model, the comparison uses the strongest available configuration from every model family. The lineup includes the tuned TF-IDF Linear SVM, the tuned Text CNN, the baseline DeBERTa-v3 and baseline GPT-2, the tuned DeepSeek-R1 with QLoRA, and the tuned TinyLLaMA with QLoRA. The primary basis is test-set performance, and for a practical context, model size and observed training time are included. Dataset and corpus evaluation are addressed separately and do not affect the rankings in this section.

Model	**Precision**	**Recall**	**F1-score**	**Accuracy**	**Model Size (MB)**	**Training Time (hr)**
SVM	0.81	0.76	0.78	0.82	12.6	1.5
CNN	0.83	0.81	0.82	0.85	31.7	1
DeBERTa-v3	0.90	0.91	0.91	0.93	720.4	22
GPT-2	0.86	0.87	0.87	0.89	486.1	1.5
DeepSeek-R1	0.87	0.87	0.87	0.90	36.1	13.3
TinyLLaMA	0.89	0.89	0.89	0.92	12.4	17.5

Table 10: Model Comparison

The table shows a clear performance gradient across model families. Obviously, transformer-type models lead in classification quality, with DeBERTa-v3 achieving the highest scores overall, which has an F1-score of 0.91 and 93% accuracy, followed closely by TinyLLaMA with 0.89 F1-score and 92% accuracy. GPT-2 and DeepSeek-R1 group in the mid-tier, with both having an F1-score of 0.87, while machine learning and deep learning models trail behind, CNN and SVM, respectively, have an F1-score of 0.82 and 0.78. This pattern aligns with the expected modelling capacity that SVM relies on sparse n-gram counts, CNN adds local phrase patterns, and transformers can capture longer-range dependencies and discourse cues, benefiting from their pretrained characteristics.

Macro average on the precision–recall balance can also be compared. DeBERTa-v3 and TinyLLaMA show near-symmetric precision and recall. This shows good class separation rather than threshold luck. DeepSeek-R1 has a small gap between 90% accuracy and 87% F1, as in earlier

confusion matrices that showed strong performance on dominant classes and softer boundaries on minority ones, which is OTH. CNN ranks over SVM by increasing recall with 0.81 and 0.76, suggesting that its structure better recovers true positives, but it still underperforms the transformer group, where nuanced context is critical.

Cost and consumed resources also introduce meaningful trade-offs. Although DeBERTa-v3 achieves the top scores, it requires the largest storage, approximately 720 MB, and has training times of up to 22 hours. TinyLLaMA also achieves high accuracy with a nearly 58 times smaller size (approximately 12.4 MB) than DeBERTa, capitalising on the benefits QLoRA brought, although training also required substantial time, approximately 17.5 hours. DeepSeek-R1 combines a small footprint with a size of about 36 MB, helped by QLoRA, with good performance, but its training time is about 13.3 hours. GPT-2 stands out for short training time, about 1.5 hours (14.7 times less than DeBERTa-v3) and a smaller size than DeBERTa-v3, about 486 MB. CNN trains fastest in about 1 hour, with a small size of 31.7 MB, and SVM remains the lightest baseline (12.6 MB and 1.5 hours training time), but the performance is weak compared to others.

Overall, through the comparison, there are various options rather than a single model when studying from multiple perspectives. DeBERTa-v3 maximises performance, TinyLLaMA has a good ratio between size and quality, DeepSeek-R1's performance is so-so with training cost slightly lower than previous model, but also has small model size, while GPT-2 is the only transformer with fastest training time, but size and performance are moderate, and lastly, CNN and SVM become reliable baselines.

Model	**Class**	**Precision**	**Recall**	**F1-score**
SVM	C	0.85	0.73	0.78
	G	0.82	0.83	0.83
	W/Q	0.83	0.91	0.87
	OTH	0.72	0.57	0.64
CNN	C	0.88	0.79	0.83
	G	0.85	0.86	0.85
	W/Q	0.88	0.92	0.90
	OTH	0.71	0.66	0.68

DeBERTa-v3	C	0.90	0.94	0.92
	G	0.94	0.94	0.94
	W/Q	0.95	0.94	0.95
	OTH	0.82	0.83	0.83
GPT-2	C	0.84	0.92	0.88
	G	0.90	0.91	0.91
	W/Q	0.94	0.91	0.92
	OTH	0.77	0.74	0.75
DeepSeek-R1	C	0.90	0.89	0.90
	G	0.89	0.93	0.91
	W/Q	0.94	0.92	0.93
	OTH	0.76	0.75	0.76
TinyLLaMA	C	0.92	0.90	0.91
	G	0.92	0.93	0.93
	W/Q	0.95	0.94	0.94
	OTH	0.79	0.80	0.80

Table 11: Class-Wise Performance Comparison

A fast review across classes, transformers dominate, while OTH remains the hardest boundary for every model. DeBERTa-v3 presents the best overall result with high precision and recall in all classes and the top F1 for C, G, and W/Q, while TinyLLaMA tracks closely behind with only small gaps. GPT-2 and DeepSeek-R1 are lower, sharing a similar pattern, and the classical baselines, SVM and CNN, are weaker, especially in OTH.

For Claim (C), performance rises steadily with model capacity. DeBERTa-v3 leads at F1 0.92 with both precision and recall at more than 0.90. TinyLLaMA follows at 0.91 and DeepSeek-R1 at 0.90. GPT-2 shows a recall-heavy profile for C with 0.92 recall and 0.84 precision, showing that it retrieves many true claims but accepts more false positives than the top models. CNN drops to 0.83 and SVM to 0.78. This shows their limited ability to perform as strongly as the transformer model.

For Grounds (G), DeBERTa-v3 again is the highest with 0.94 F1-score and the same precision and recall. TinyLLaMA is always close at 0.93, while DeepSeek-R1 and GPT-2 land at 0.91. The CNN (0.85) improves over SVM (0.83) by recovering more true cases, but still misses the nuanced patterns that transformers pick up. The main error trend found in earlier matrices is that G is being pulled into W/Q, which shrinks as models get stronger.

For Warrant or Qualifier (W/Q), all models perform well. DeBERTa-v3 still leads with an F1-score of 0.95, TinyLLaMA is at 0.94, DeepSeek-R1 at 0.93, and GPT-2 at 0.92. While CNN and SVM are also good at 0.90 and 0.87. Precision and recall are well-balanced for the best models.

For Others (OTH), weakness is a universal phenomenon which can be directly linked to the dataset corpus. DeBERTa-v3 achieves the strongest F1 at 0.83 with balanced precision and recall, TinyLLaMA follows at 0.80, DeepSeek-R1 at 0.76, GPT-2 at 0.75, CNN at 0.68, and SVM at 0.64. OTH sentences are heterogeneous and often border on light evidence or implicit reasoning. This explains both false positives in W/Q and lower recall for OTH itself. This class is the main source of remaining error.

In summary, class-wise results show a consistent hierarchy. DeBERTa-v3 is the best across all categories among all models, TinyLLaMA provides near-best scores with slightly softer OTH handling, DeepSeek-R1 and GPT-2 have mid-tier performance with small trade-offs in precision or recall depending on class, and CNN and SVM become the baselines. Accordingly, DeBERTa-v3 is selected for deployment on the strength of its classification performance, its model size, and training cost are acceptable within the project's constraints, so it is not decisive.

Author	**Dataset**	**Task (classes)**	**Model**	**Result (macro F1)**
Stab & Gurevych (2017)	402 English essays from essayforum.com	ATC — Major Claim, Claim, Premise	Joint ILP (feature-rich SVM/CRF)	0.826
Niculae et al. (2017)	UKP dataset	ATC — Major Claim, Claim, Premise	Structured SVM	0.776

Mushtaq & Cabessa (2023)	Persuasive Essays by Stab and Gurevych	ATC — Major Claim, Claim, Premise	BERT–MINUS–FeaTxt–Auto	0.831
Cabessa et al. (2024)	Persuasive Essays by Stab and Gurevych	ATC — Major Claim, Claim, Premise	GPT-3.5–FeaTxt (fine-tuned, 2 ep.)	0.863
Favero et al. (2025)	Feedback Prize – Predicting Effective Arguments (PERSUADE 2.0 subset)	ATC — Lead, Position, Claim, Counterclaim, Rebuttal, Evidence, Concluding Statement	GPT-4o mini Llama 3.1 8B	0.848 0.515

Table 12: Previous Studies Discussion

The related work table shows an increase in macro F1 as methods move from structured SVM pipelines to modern transformer and small LLM approaches. On the Persuasive Essays (PE) benchmark with three classes, Major Claim, Claim, and Premise, early joint systems such as Stab and Gurevych report 0.826 macro F1. Modular and LLM-based setups increase, with BERT MINUS FeaTxt Auto at 0.831 and GPT 3.5 with features as text at 0.863. A separate education dataset with more labels shows mixed outcomes, where GPT 4o mini reaches 0.848 while Llama 3.1 8B records 0.515.

In this study, DeBERTa v3 has the best result, reaching macro F1 of 0.91 for sentence-level classification across Claim, Grounds, Warrant or Qualifier, and Others. Although these accuracies and macro F1 are higher than the numbers reported in the three-class essay studies, they are meaningless for comparison and act as an indicator. The dataset differs in domain and label inventory. There is no prior work that uses the IBM Debater transcripts for this ATC task, and the corpus annotations in this project are not perfect, which can raise uncertainty in both the ceiling and comparability of the scores.

Overall, the evidence points to clear gains from modern transformers, but the present findings should be treated as internally valid only. Stronger external validity will require testing the same label definitions on a public benchmark, such as Persuasive Essays, or releasing a mapped subset of IBM Debater with transparent guidelines and inter-annotator agreement, then rerunning the best

pipeline. Two areas are most likely to improve results, including better handling of the Others class and adding limited context beyond single sentences.

In conclusion, the comparison in this section shows a consistent hierarchy across model families. Among the best configurations, DeBERTa v3 shows the best test performance on this debate dataset, TinyLLaMA trails narrowly while offering an excellent size and deployment profile, GPT 2 and DeepSeek R1 form a competitive middle, and the classical SVM and CNN baselines, though faster to train, remain behind on macro F1. Class-wise results are stable across families. Warrant or Qualifier is the easiest boundary, Claim and Grounds benefit most from transformer context, and Others is universally the hardest class, with residual errors concentrated in confusions from Others into Warrant or Qualifier and Grounds. These patterns align with the literature, where performance improves as methods transition from structured SVM pipelines to modern transformers and small LLMs, while also reflecting the long-standing challenge of handling non-canonical or heterogeneous segments. Because datasets, label inventories, and segmentation regimes are different across studies, direct score matching is not valid. Transformer fine-tuning provides the most reliable balance of precision and recall for sentence-level argument typing, and compact adapters can approach that quality with lower footprints.

5.5 Model Deployment

Debate Analyser will be deployed as a lightweight Streamlit web application. Within its operation, a punctuation model firstly cleans the input transcript. The chosen model (DeBERTa-v3) starts labelling each sentence as Claim, Ground, Warrant/Qualifier, or Others, while a Hugging Face fallacy detector points out common fallacies with an adjustable threshold. The result will be presented in visualised charts, along with recommendations based on debate flow and fallacies for improvement, generated by Gemini AI. The interface is designed to be simple for users. The sources can be visited via:

- Debate Analyser: https://debate-analyzer.streamlit.app/
- Deployment Source Package: https://github.com/Sauyang520/debate-analyzer
- ATC Model: https://huggingface.co/Sauyang/argument-labelling-deberta-v3

5.5.1 Preparation and Deployment

The debate analyser dashboard development begins with programming for an interface with planned functions and mechanisms. The main components of the debate analyser include:

```
# ==== Input section ====
col1, col2 = st.columns(2)
with col1:
    st.subheader("Input Transcript")
    typed_text = st.text_area("Transcript:", placeholder="Paste or type
here...")
with col2:
    st.subheader("or Upload TXT File")
    uploaded_file = st.file_uploader("Upload a .txt file", type=["txt"])

# Threshold selector
st.subheader("Fallacy Detection Settings")
st.session_state.fallacy_threshold = st.slider(
    "Fallacy Score Threshold",
    min_value=0.0, max_value=1.0, step=0.05,
    value=0.8, format="%.2f"
)

candidate_text = ""
if uploaded_file:
    try:
        candidate_text = uploaded_file.read().decode("utf-8").strip()
    except UnicodeDecodeError:
        candidate_text = uploaded_file.read().decode("latin-1").strip()
elif typed_text:
    candidate_text = typed_text.strip()

if st.button("Analyze", type="primary", disabled=not bool(candidate_text)):
    st.session_state.analysis_text = candidate_text
    st.session_state.show_results = True
    st.rerun()
```

Code Snippet 83: Create an Input Box for the Debate Transcript

Firstly, the script builds the dashboard's intake and trigger logic. It offers two ways to provide data, including typing or pasting a transcript, or uploading a .txt file. Then, it exposes a single control for fallacy sensitivity via a threshold slider, allowing users to decide how strict the detector should be. The code reads the uploaded file with a fallback decoding method, then assembles the chosen input as a single candidate text. When the "Analyse" button is pressed, the transcript and a "show results" flag are stored in session state, and the app refreshes.

```
@st.cache_resource(show_spinner=False)
def load_punct_model():
    from deepmultilingualpunctuation import PunctuationModel
    return PunctuationModel()



# ==== Constants ====
VALID_LABELS = ['C', 'G', 'W/Q', 'OTH']
IDX2LABEL = {i: lab for i, lab in enumerate(VALID_LABELS)}

# ==== Processing ====
if st.session_state.show_results and st.session_state.analysis_text:
    raw_text = st.session_state.analysis_text
    current_hash = hashlib.md5(raw_text.encode("utf-8")).hexdigest()

    if st.session_state.text_hash != current_hash or not
st.session_state.restored_text:
        with st.spinner("Restoring punctuation…"):
            restored = load_punct_model().restore_punctuation(raw_text)
        st.session_state.restored_text = restored
        st.session_state.text_hash = current_hash

    final_text = st.session_state.restored_text
    threshold = st.session_state.fallacy_threshold

    st.markdown("---")
    st.header("Results")

    sentences = nltk.sent_tokenize(final_text, language="english")
    st.write(f"Your debate transcript contains {len(sentences)} sentences!")
```

Code Snippet 84: Process Input Transcript

After that, the script prepares and runs the first stage of processing after the user clicks the "Analyse" button. The punctuation restorer is loaded once and cached, so the heavy model does not reload on every rerun. When results are requested, the app grabs the input text from the session state. The deep-multilingual punctuation model is called to reconstruct commas and periods, then saves the restored text back to the session for reuse. Simple constants define the four argument labels and an index-to-label map for later classification. NLTK is used to split the restored text into sentences and report how many were found.

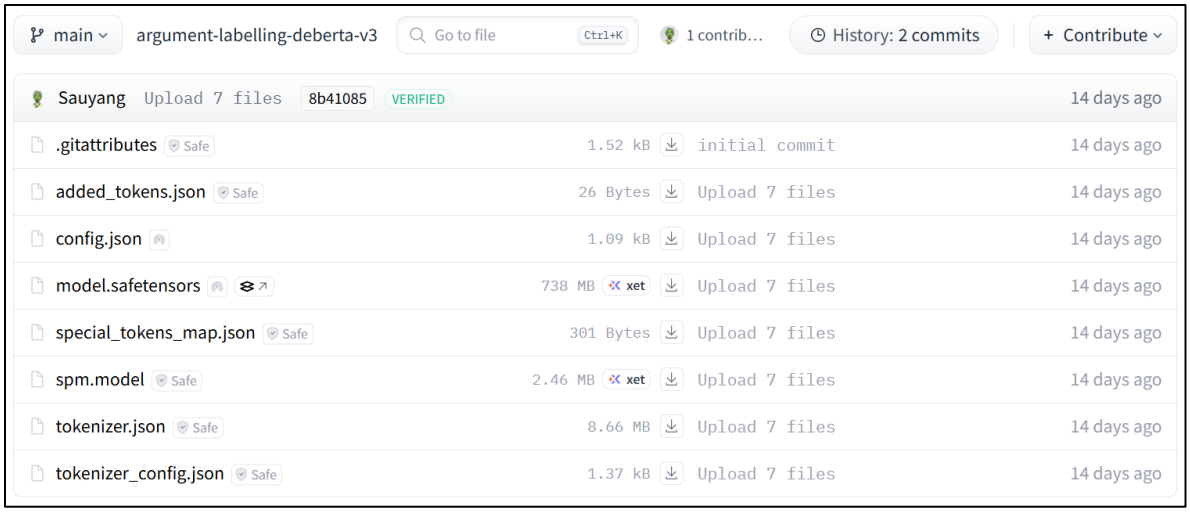


Figure 112: Hugging Face Repository for Argument Type Classifier

Then, the chosen classifier model is published to the Hugging Face Hub because it natively supports large model artefacts via Git-LFS, while GitHub's standard limit is 25 MB, and the DeBERTa-v3 checkpoint is about 720 MB. The repository stores all important files of the model and the tokeniser in one place. Hosting on the Hub also adds a model card, license metadata, and an integrity "VERIFIED" badge, along with a global CDN and local caching, so the Streamlit app fetches the model once and then runs offline.

```
@st.cache_resource(show_spinner=False)
def load_arg_classifier():
    from transformers import AutoTokenizer, AutoModelForSequenceClassification
    model_name = "Sauyang/argument-labelling-deberta-v3"
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForSequenceClassification.from_pretrained(model_name)
    model.eval()
    if torch.cuda.is_available():
        model.cuda()
    return tokenizer, model



with st.spinner("Classifying argument types…"):
    tokenizer, model = load_arg_classifier()
    preds, confs = [], []
    batch_size = 16

    def to_device(batch):
        return {k: v.cuda() for k, v in batch.items()} if
torch.cuda.is_available() else batch

    for i in range(0, len(sentences), batch_size):
        enc = tokenizer(
            sentences[i:i+batch_size],
            padding=True, truncation=True,
            return_tensors="pt", max_length=256
        )
        enc = to_device(enc)
        with torch.no_grad():
            probs = softmax(model(**enc).logits, dim=1)
            preds.extend(torch.argmax(probs, dim=1).cpu().tolist())
            confs.extend(probs.max(dim=1).values.cpu().tolist())

arg_labels = [IDX2LABEL[i] for i in preds]
arg_conf = [round(c, 4) for c in confs]
```

Code Snippet 85: Run DeBERTa-v3 Argument Type Classification Model

Then, the fine-tuned DeBERTa-v3 argument classifier is loaded from the Hugging Face Hub, and it is cached so it only loads once. When the user requests results, the app tokenises the transcript sentences and runs them through the model in small batches to keep the interface responsive. For each sentence, it outputs a predicted argument role and a confidence score derived from the model's probabilities. Finally, the numeric predictions are converted into the four readable labels, including C, G, W/Q, and OTH.

```
@st.cache_resource(show_spinner=False)
def load_fallacy_pipeline(
    model_src: str = "q3fer/distilbert-base-fallacy-classification",
    prefer_local_dir: str = "models/fallacy"
):
    from transformers import pipeline
    from pathlib import Path

    if Path(prefer_local_dir).is_dir():
        return pipeline(
            "text-classification",
            model=str(prefer_local_dir),
            tokenizer=str(prefer_local_dir),
            use_safetensors=True,
            device=0 if torch.cuda.is_available() else -1
        )
    return pipeline(
        "text-classification",
        model=model_src,
        tokenizer=model_src,
        device=0 if torch.cuda.is_available() else -1,
        torch_dtype=None
    )

with st.spinner("Detecting logical fallacies…"):
    fallacy_pipe = load_fallacy_pipeline()
    fallacy_results = fallacy_pipe(
        sentences, truncation=True, padding=True, max_length=256, batch_size=16
    )

def _pick(res):
    if isinstance(res, list):
        res = res[0]
    return res["label"], float(res["score"])

fallacy_labels, fallacy_scores = zip(*[_pick(r) for r in fallacy_results]) if
sentences else ([], [])
```

Code Snippet 86: Run Fallacies Detector Model

The fallacy-detection stage is set up and run. It initialises a Hugging Face text-classification pipeline, and caching ensures the detector loads only once. During analysis, the pipeline processes all sentences in batches and returns, for each sentence, a detected fallacy type and a confidence score. A small helper then standardises those outputs into two aligned lists of labels and scores. The user's threshold later filters these raw predictions with a default of 0.8.

```
with r1c1:
    fig_wc, ax_wc = plt.subplots(figsize=FIGSIZE_UNI)
    wc = WordCloud(width=900, height=360, background_color="white",
                   stopwords=STOPWORDS).generate(final_text)
    ax_wc.imshow(wc, interpolation="bilinear")
    ax_wc.axis("off")
    ax_wc.set_title("Word Cloud", fontsize=12, pad=6)
    st.pyplot(fig_wc, use_container_width=True)
```

Code Snippet 87: Insert Visualisation for Transcript Analysis

The script above will present a simple word cloud chart in the dashboard. The figure is created with Matplotlib. It is one of the six visuals that highlight dominant terms and themes briefly.

```
GEMINI_API_KEY = st.secrets["gemini"]["api_key"]
genai.configure(api_key=GEMINI_API_KEY)

advice = ""
def get_gemini_recommendations(df, model_name="gemini-2.0-flash"):
    summary = []
    for _, row in df.iterrows():
        summary.append(f"Sentence: {row['sentence']}\n"
                       f"Argument Type: {row['argument_type']}\n"
                       f"Fallacy: {row['fallacy_final']}\n")
    summary_text = "\n".join(summary[:50])

    prompt = f"""
    You are an experienced debate coach. Review the transcript with identified
argument types and fallacies.

    Instructions:
    - Focus only on the weaknesses and issues in this transcript.
    - Give direct, professional, and actionable advice to improve (the flow,
style, content, etc).
    - Be concise and critical: use short sentences or bullet points.
    - Avoid general textbook explanations, only refer to the transcript
problems.
    - Maximum 6 bullet points

    Transcript analysis:
    {summary_text}
    """

    model = genai.GenerativeModel(model_name)
    response = model.generate_content(prompt)
    return response.text if response else "No recommendations generated."

if GEMINI_API_KEY:
    st.subheader("Gemini Flash 2.0 Recommendations")
    with st.spinner("Generating recommendations..."):
        advice = get_gemini_recommendations(df)
    st.write(advice)
```

Code Snippet 88: Connect and Prompt Gemini 2.0 Flash

This script adds an automated feedback step powered by Google's Gemini. The app securely reads the API key, then, after argument labels and fallacy flags are prepared in a dataframe, it compiles a summary of up to 50 sentences containing each sentence with classified argument type, and any detected fallacy. That summary is sent to the gemini-2.0-flash model with a tightly scoped prompt that asks for at most six bullet points, focused only on weaknesses in the given transcript and giving direct, actionable fixes rather than generic theory.

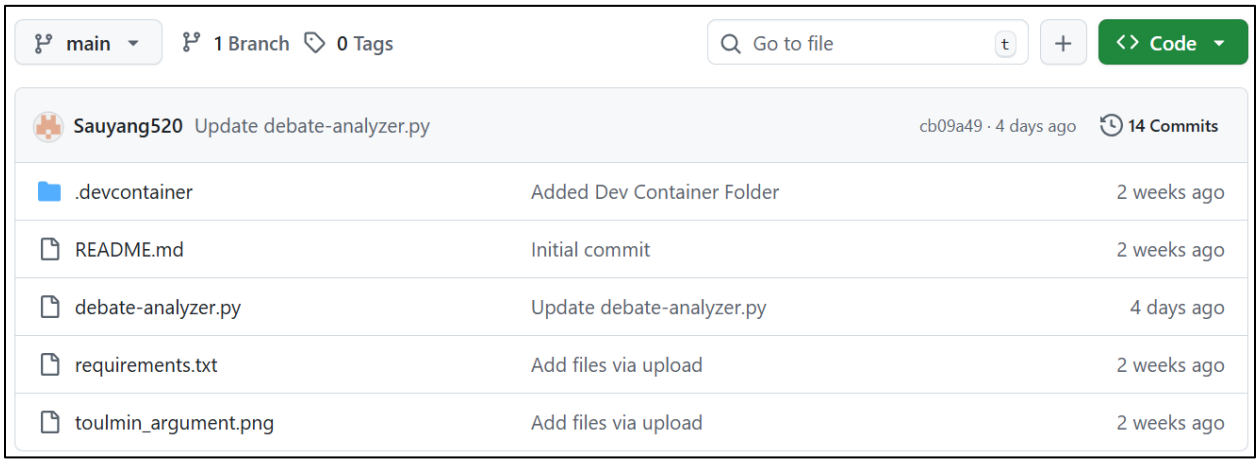


Figure 113: GitHub Repository for Debate Analyser

After the script for the debate analyser is completed, it is uploaded to GitHub, along with a requirements text file that stores all the necessary libraries. All setups are done, and the debate analyser is ready to be deployed.

Figure 114: Debate Analyser Deployment

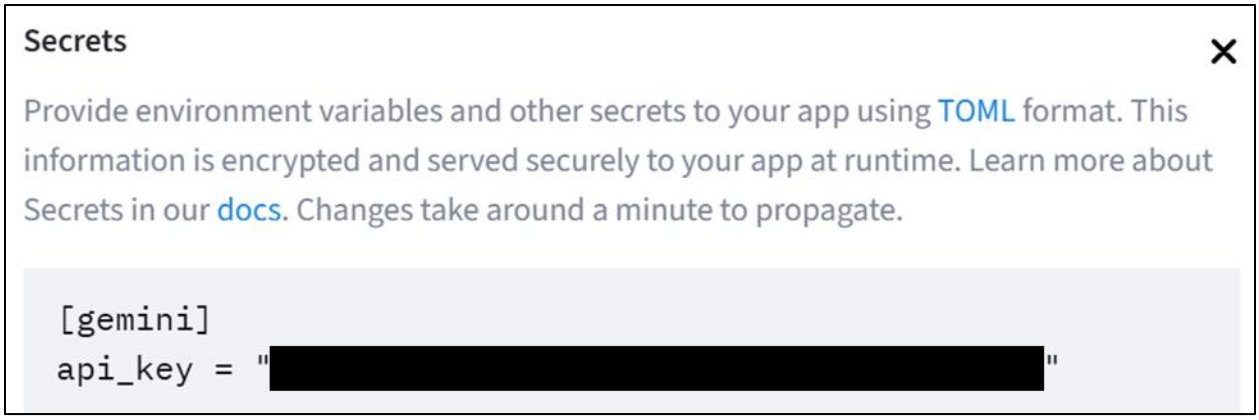


Figure 115: Secrets Setup for Gemini API Key

Deployment on Streamlit Community Cloud is quick and straightforward by linking to the targeted GitHub repository project. Sensitive credentials, such as the Gemini API key, are stored in the app's Secrets (TOML) and read at runtime, keeping them out of the codebase and preventing accidental exposure.

5.5.2 Debate Analyser System Overview

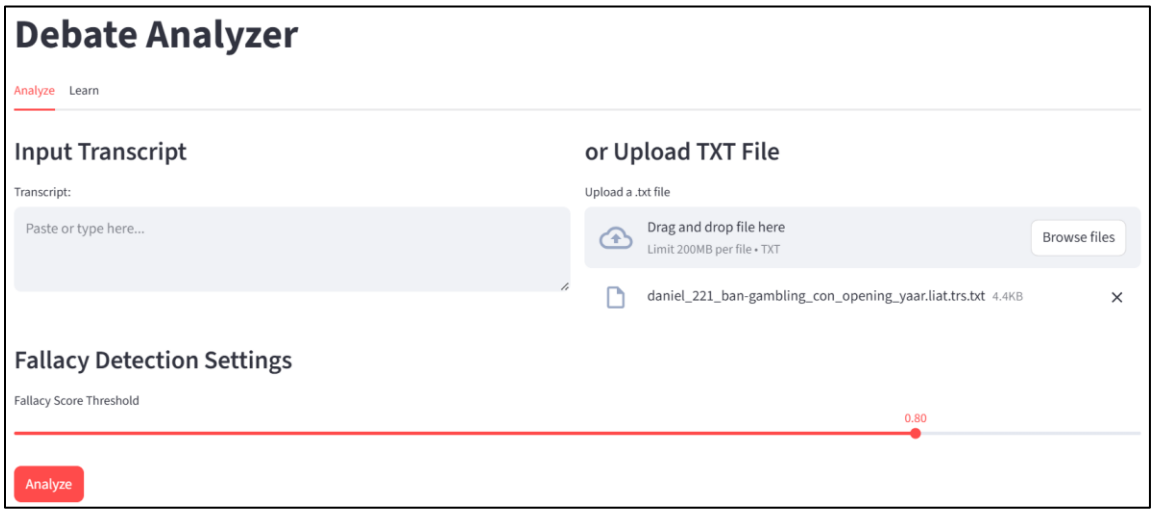
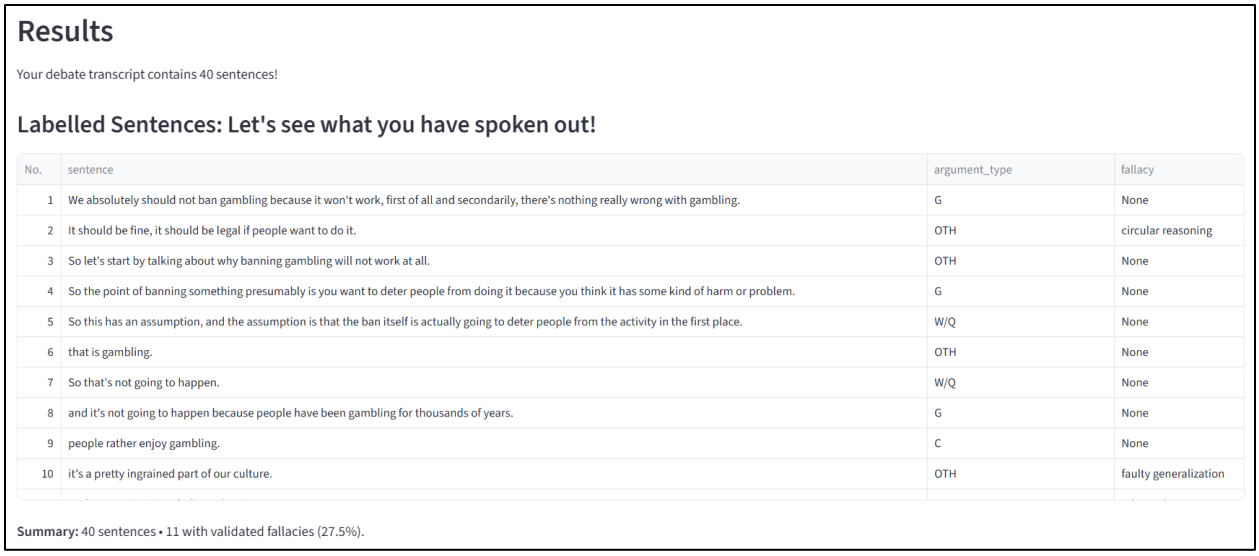


Figure 116: Debate Analyser Input Section

The debate analyser is designed to be clean and straightforward. Once the page is loaded, users can paste a transcript or upload a TXT file, then choose how strict the fallacy detector should be using the score threshold slider. The "Analyse" button will trigger the whole pipeline for analysis.



Results

Your debate transcript contains 40 sentences!

Labelled Sentences: Let's see what you have spoken out!

No.	sentence	argument_type	fallacy
1	We absolutely should not ban gambling because it won't work, first of all and secondarily, there's nothing really wrong with gambling.	G	None
2	It should be fine, it should be legal if people want to do it.	OTH	circular reasoning
3	So let's start by talking about why banning gambling will not work at all.	OTH	None
4	So the point of banning something presumably is you want to deter people from doing it because you think it has some kind of harm or problem.	G	None
5	So this has an assumption, and the assumption is that the ban itself is actually going to deter people from the activity in the first place.	W/Q	None
6	that is gambling.	OTH	None
7	So that's not going to happen.	W/Q	None
8	and it's not going to happen because people have been gambling for thousands of years.	G	None
9	people rather enjoy gambling.	C	None
10	it's a pretty ingrained part of our culture.	OTH	faulty generalization

Summary: 40 sentences • 11 with validated fallacies (27.5%).

Figure 117: Processed Transcript with Labelled Argument Types and Fallacies

After processing, the app restores punctuation, segments the transcript into sentences, and classifies each one as Claim → C, Grounds → G, Warrant or Qualifier → W/Q, or Others → OTH. A fallacy detector then flags any logical fallacies that exceed the user-selected threshold. The outputs are shown in an interactive table listing the sentence index, text, predicted argument type, and any validated fallacy, followed by a summary of totals and fallacy rate.

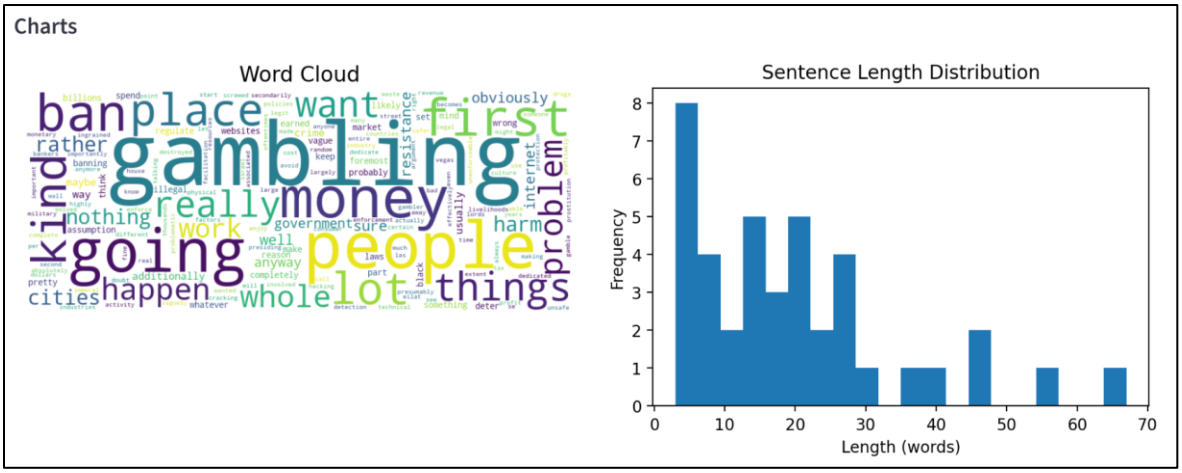


Figure 118: Word Cloud and Sentence Length Distribution

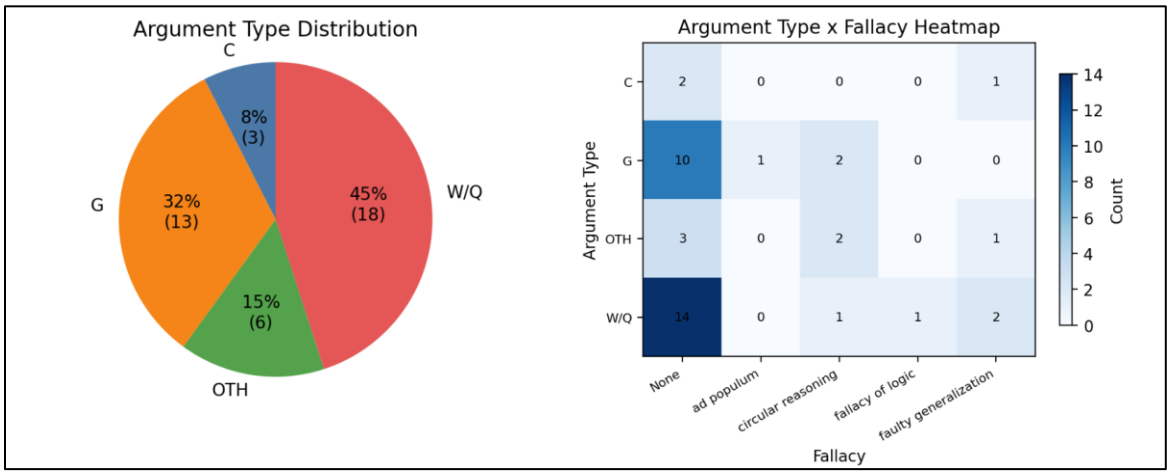


Figure 119: Argument Type Distribution with Fallacy Heatmap

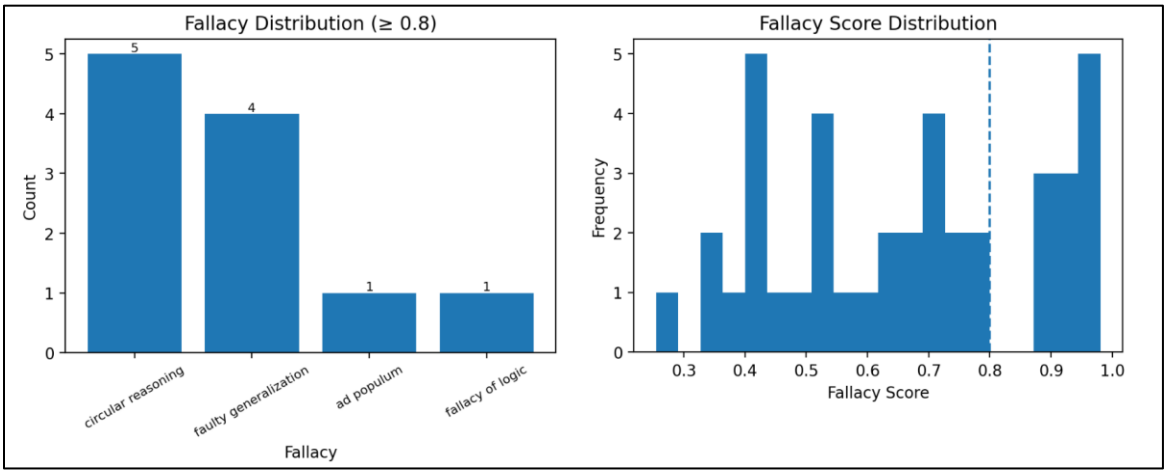


Figure 120: Fallacy and Its Score Distribution

Then, the dashboard summarises the transcript visually, so users can quickly identify where to revise. The word cloud reveals dominant topics and off-topic drift, the sentence-length histogram shows pacing and clarity, signalling when arguments are too choppy or overly long. The argument-type pie chart highlights balance across Claim, Grounds, Warrant or Qualifier, and Others, making gaps obvious. The heatmap links each argument type to detected fallacies, helping pinpoint patterns. Finally, the fallacy bar chart and score histogram show which fallacies dominate above the chosen threshold and how confident the detector is, which guides whether to tighten or relax the setting.

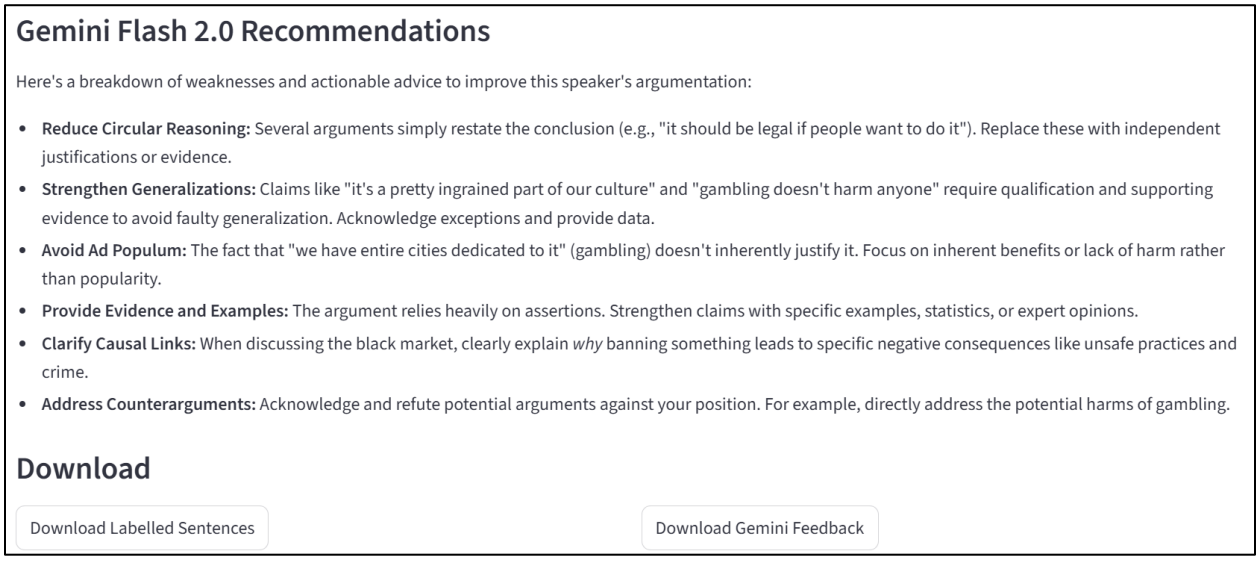


Figure 121: Recommendations by Gemini

Finally, the dashboard presents a “Gemini Flash 2.0 Recommendations” panel that converts the model’s findings into action-oriented recommendations. The bullets focus only on weaknesses surfaced by the analysis, so users know exactly what to revise next. Below the advice, two export buttons allow users to download the labelled-sentence table and the Gemini feedback for record-keeping and grading purposes. Overall, this closes the loop from raw transcript to actionable guidance and take-away artefacts that support learning and improvement.

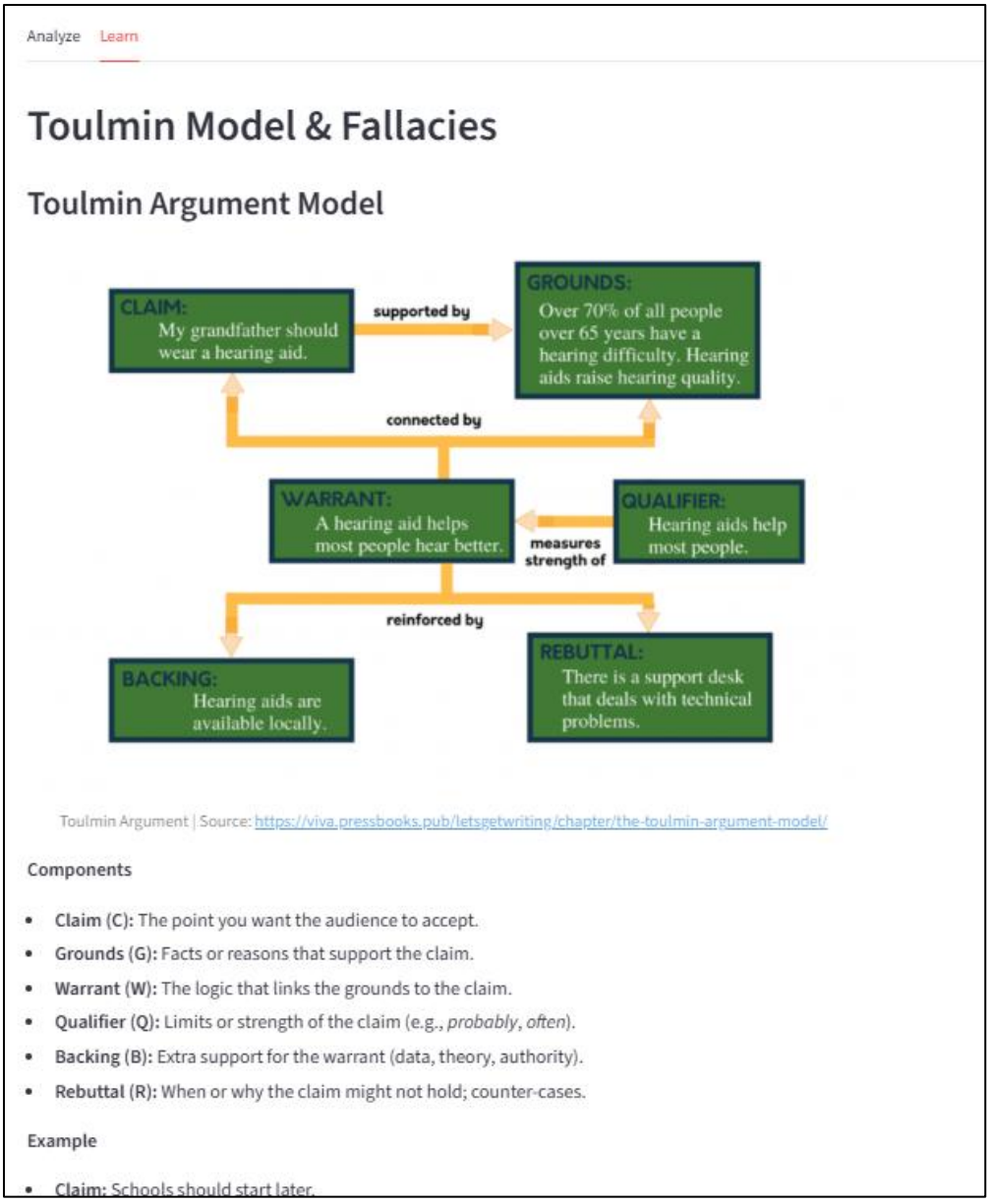


Figure 122: Debate Analyser Learning Tab

The debate analyser also includes a Learn tab that gives users a quick, practical primer on the Toulmin model and common logical fallacies. It presents a simple visual representation of the framework and explains each component, including Claim, Grounds, Warrant, Qualifier, Backing, and Rebuttal, along with a brief example for context. Below that is a fallacy guide that defines frequent errors, such as ad populum, circular reasoning, and faulty generalisation. This helps users understand the labels the system outputs.

5.6 Summary

In summary, this chapter evaluates all models on the debate dataset, reports the main metrics, and discusses the implications of the errors. After tuning hyperparameters on a 100-transcript sample and retraining on the full corpus, transformer models clearly outperform classical baselines, with DeBERTa-v3 achieving the strongest overall results and cleaner ROC curves and confusion patterns, while SVM and CNN remained useful as lightweight references. Class-wise analysis shows that W/Q is the most separable category, C and G benefit from longer context signals, and OTH is consistently the hardest boundary, often being confused with light evidence or implied reasoning. The tuned DeepSeek model shows these trends by reducing over-prediction of W/Q and improving recall for C and G. The model comparison also considered efficiency, noting that the best accuracy comes with larger models and longer training. In contrast, smaller adapters and compact models have an attractive size and deployment profiles. A brief look at prior studies on essay corpora confirms the same broad pattern that modern transformers lift argument type classification. However, direct score matching is not claimed due to different datasets, labels, and segmentation choices. Overall, Chapter 5 establishes a clear performance hierarchy and identifies where the remaining errors concentrate.

Chapter 6: Conclusion

6.1 Critical Evaluation

6.1.1 Project Achievements

The project achievements can be validated through the project objectives:

1. To collect and compile a comprehensive dataset of debate transcripts, including sentence-level argument type labelling and fallacy annotations.

In this project, IBM Project Debater transcripts were consolidated, deduplicated, and cleaned; punctuation was restored to fix run-on speech, and then transcripts were segmented into sentences with additional, rule-based discourse cue splitting for complex cases. A stratified 5% sample was annotated at the sentence level using a simplified Toulmin scheme (Claim, Grounds, Warrant or Qualifier, Others) with assisted labelling and human verification. A tuned DeBERTa transfer-learning model then propagated labels to the whole corpus, followed by a targeted manual revision pass on 7,060 sentences to improve quality. A pretrained fallacy detector was integrated and filtered with a 0.7 confidence threshold to keep only reliable fallacy tags. A corpus-level label-quality estimate (about 82.5%) was computed to make downstream evaluation transparent. Remaining weaknesses include heterogeneity in the Others class and the fact that the fallacy labels come from an external model rather than a task-specific, fine-tuned detector.

2. To train and fine-tune models to automatically identify argument types and adopt the model to detect logical fallacies from text.

This objective is achieved when multiple model families were implemented under a consistent pipeline: TF-IDF Linear SVM, Text CNN, and four transformers (DeBERTa-v3, GPT-2, DeepSeek-R1 with QLoRA, TinyLLaMA with QLoRA). Transformer hyperparameters were tuned on a 100-debate subset, and the best settings were retrained on the full data. DeBERTa-v3 emerged as the strongest configuration, reaching accuracy around 0.93 and macro F1 around 0.91 on the debate test split, with steady ROC behaviour and reduced cross-class leakage among Claim, Grounds, and Warrant or Qualifier. The pretrained fallacy detector was adopted rather than retrained.

3. To assess and compare the performance of different models using standard evaluation metrics.

To achieve this objective, all models were evaluated using accuracy, precision, recall, and macro F1, supported by class-wise reports, confusion matrices, and ROC curves. A comparison table summarised effectiveness and efficiency (model size and training time) to highlight practical trade-offs. Findings are consistent across views: transformers outperform classical baselines; Warrant or Qualifier is generally the easiest boundary; Claim and Grounds benefit most from contextual modelling; Others remains the hardest category and the main source of residual errors. Results were interpreted alongside the corpus-quality estimate to avoid overstating absolute scores and to separate model behaviour from annotation noise.

4. To develop and deploy an interactive system that analyses input transcripts, detects fallacies, identifies argument types, and provides suggestions for improvement.

A Streamlit application was built that accepts raw transcripts, performs cleaning and segmentation, classifies sentence-level argument roles, attaches high-confidence fallacy flags, and presents results with per-class confidence and concise summaries. The system also generates actionable suggestions to strengthen weak segments using Gemini Flash 2.0.

In summary, this project demonstrates a complete pipeline from debate data and transparent labelling, through tuned modelling and evaluation, to a usable tool that delivers structured, formative feedback on argument quality. This outcome directly supports Sustainable Development Goal 4: Quality Education by making feedback accessible at scale, helping learners practise critical thinking and clearer reasoning, and giving educators a practical aid for consistent assessment. By highlighting which sentences function as Claim, Grounds, Warrant or Qualifier, and Others, the system encourages targeted revision and self-reflection, reduces reliance on one-off manual comments, and promotes more equitable learning support across different proficiency levels.

6.1.2 Personal Development and Skill Gains

This project has been a long and steady exercise in learning how to think and work effectively. Knowledge from academic study, Research Methods for Computing and Technology (RMCT), guided the early design of the study, from setting clear objectives to selecting methods that match those objectives. The knowledge of how to collect evidence and write in a structured way came directly from RMCT. Data Mining and Predictive Modelling (DMPM) and Text Analysis and Sentiment Analysis (TXSA) helped convert ideas into a repeatable workflow for text data, including cleaning, segmentation, tokenisation, and building baselines. Optimisation and Deep Learning (ODL), which is studied during the semester of FYP implementation, supported the transition from traditional models to deep learning by introducing concepts such as optimisation, regularisation, and convergence. Together, these modules formed a step-by-step approach to working that was crucial for the researcher, who learns slowly but steadily and requires time to understand each part before moving on.

The study also required learning beyond the taught curriculum. Transformers, pre-trained language models, and transfer learning became core skills because debate transcripts do not behave like small tabular datasets used in many assignments. The corpus was large, unlabelled, and messy in places, so the researcher had to design an annotation path, define a simple Toulmin scheme, and combine assisted labelling with human checks. There was no shortcut for this part. It demanded reading, small pilots, and many adjustments to the rules for sentence boundaries and discourse cues. Understanding QLoRA, adapter tuning, and how to fit training into a modest GPU budget came later, but only after the basics of data quality and label consistency were in place.

Model building shifted the researcher from a general push for higher numbers to a resource-aware and evidence-driven approach. Because full fine-tuning of large pretrained models consumed too much time, and the available GPU and memory were not sufficient, the researcher needed to think of a practical strategy for adoption. A development set of one hundred debates was sampled to search for sensible hyperparameters first, and only the best setting was then retrained on the full dataset. This workflow reduced waste and kept experiments reproducible. Along the way, the researcher moved beyond the idea that higher accuracy always means a better model. Attention shifted to macro F1, class-wise precision and recall, and signs of overfitting, such as a widening gap between training and validation or unstable metrics across folds. Reading confusion matrices

and ROC curves also helped locate exactly where errors occur and ensured that final choices reflect balanced performance rather than a single headline score. Working with a difficult class like Others was a lesson in humility. The model often confused incidental sentences with light evidence or implied reasoning, indicating that some problems stem from the label design, not just the algorithm. The researcher learned to question labels, to estimate corpus quality, and to report uncertainty. This mindset helped reduce overclaiming and led to clearer writing about the actual meaning of the results.

The researcher's soft skills also improved alongside the technical ones. There were many moments when the work felt too impossible. Friends suggested switching to an easier topic; early attempts at scraping YouTube debates produced low-quality transcripts, and the first training runs failed or yielded strange results. The main lesson was to keep going. The researcher learned to break problems into smaller pieces, to read and try again, and to accept that the first version would not be the last. Communication also improved through regular writing, careful tables, cleaner figures, and consistent citation practice, which made feedback from supervisors easier to act on. Keep trying and keep trying, no matter whether the way is easy or hard.

In the end, this project altered the researcher's approach to challenging tasks. It demonstrated that strong results stem from patient groundwork, honest evaluation, and numerous small iterations. The final system is not perfect, and there is still much to improve, but the journey has built real capability. The researcher can now design a study, prepare complex text data, fine-tune modern models under constraints, analyse errors with care, and ship a working tool. These are durable skills that transfer to future work in natural language processing and applied machine learning, which the FYP result cannot give.

6.1.3 Potential Contribution

As this project stresses Sustainable Development Goal 4 on Quality Education, its first contribution is an accessible debate analyser that supports formative learning. The system turns unstructured transcripts into clear feedback by marking sentences as C, G, W/Q, or OTH, and it can also flag potential fallacies with concise suggestions for improvement. This structured guidance helps learners practise reasoning more often and with better focus, while providing educators with a consistent aid for reviewing large volumes of work in a limited amount of time and with limited hardware.

A second contribution is methodological. The study applies IBM Project Debater transcripts to sentence-level argument type classification, where prior work has focused mostly on essays with three labels. This project features a four-class setup, documented for formal debates. Although these four classes are not a complete Toulmin inventory, they form a pragmatic subset that matches debate flow and keeps annotation feasible at scale. End-to-end results are reported, including accuracy, macro F1, per-class metrics, confusion matrices, and ROC curves. This creates a reference for future work to move beyond essays into spoken or turn-based discourse, and the pipeline and evaluation can be reused to establish baselines and support fairer comparisons.

On the data side, the project contributes a transparent labelling pathway for debates. A simple Toulmin scheme is defined, assisted labelling is combined with human verification, and a corpus quality estimate is reported so that model behaviour is not confused with annotation noise. Comparative findings add another layer. Results confirm that modern transformers deliver the best balance of precision and recall for C, G, and W/Q, while carefully tuned compact models can approach larger systems. This gives practitioners options along a spectrum. If top accuracy is needed, a larger model can be selected. If portability and cost are the primary concerns, a smaller adapter-based or compact model can be deployed with a modest drop in quality.

Finally, the project bridges academic research and practical teaching. It connects argument mining methods to speaking tasks, offering a replicable path for building and evaluating models on debate data and packaging the outcome in a usable interface. All of these elements contribute to SDG 4 by making high-quality feedback more accessible, supporting the development of critical thinking and clear communication, and enabling more equitable access to learning support.

6.1.4 Project Strength

A major strength is the complete end-to-end pipeline. The work begins with curated debate transcripts, continues through transparent sentence labelling, and ends with a deployed tool that people can use. Each stage is documented, thanks to the FYP guideline, from cleaning and segmentation to model training and evaluation, which makes the process traceable and easier to improve in future iterations.

Another strength is methodological rigour. Performance is not presented as a single headline number. Accuracy, macro F1, and class-wise precision and recall are all reported, together with confusion matrices and ROC curves. These views make strengths and weaknesses visible at the class level. A corpus quality estimate is also provided so that readers can separate model behaviour from annotation noise. Clear reporting raises trust in the findings. Modelling choices also offer a good balance between quality and practicality. Modern transformers deliver the best scores on this debate corpus, while compact and adapter-based variants are explored for smaller footprints. This creates usable options under different resource constraints without sacrificing too much accuracy. The system is designed to have a learning impact rather than focusing solely on research metrics. The interface highlights sentence roles, shows per-class confidence, and offers concise suggestions that guide revision. This design supports formative assessment and aligns with Sustainable Development Goal 4 on Quality Education by making structured feedback accessible in settings with limited time and hardware.

Finally, the study adds value to the research community. It applies argument type classification to formal debates with a four-class scheme, reports end-to-end results that are easy to compare, and positions the outcomes against related studies without overstating cross-dataset claims. The combination of openness about limitations, careful evaluation, and a working application forms a contribution.

6.2 Limitations

For limitations, firstly, the dataset limits the project's strengths and its boundaries. The material comes from IBM Project Debater, a formal, English debate domain with well-structured speaking turns. This style differs from classroom discussions, social media threads, or mixed-language meetings, so generalisation outside formal debates is uncertain. Even within this domain, punctuation and sentence boundaries can be hard to determine because spoken language includes fillers, repairs, and long clauses. The pipeline restores punctuation and applies rule-based splitting to improve segmentation, yet a small amount of sentence fragmentation or merging remains, which can blur labels and predictions.

Labels are also another constraint. The original transcripts have no gold annotation, so sentence labels were produced through assisted labelling with ChatGPT, followed by human verification, without expert adjudication or formal inter-annotator agreement. Transfer learning then propagated labels to the rest of the corpus, which is efficient but not perfectly accurate. The label set uses four classes: Claim, Grounds, Warrant or Qualifier, and Others, rather than a full Toulmin inventory, which limits granularity for roles such as backing or rebuttal. Class frequencies are imbalanced, with Warrant or Qualifier more common and Others heterogeneous, and this skew can bias learning and depress recall for the hard classes.

Model and training choices reflect resource limits. The system operates at the sentence level, but debates rely heavily on flow across adjacent sentences and speaker turns. Without short context windows or explicit link prediction, the model can mistake context-light evidence for reasoning and miss connections between components. Hardware constraints prevented the use of the latest long-context or larger models and required QLoRA adapters for compact fine-tuning. Epoch counts, sequence lengths, and batch sizes were restricted; hyperparameters were searched on a 100-debate development set and then retrained on the full corpus. This strategy is practical under constraints but may not find globally optimal settings.

In addition, fallacy detection is a provisional component. It relies on an external model that was not fine-tuned on the debate corpus and that covers a limited set of fallacy types. A single confidence threshold controls what users see, which can introduce false positives for borderline rhetoric and false negatives for rare fallacies. Because the main focus of evaluation was argument

type classification, the end-to-end accuracy of the fallacy module remains under-measured. It should be treated as indicative feedback rather than definitive judgment.

Finally, evaluation and deployment are narrow in scope. Results are reported on one debate source without cross-dataset tests, user studies, or educator trials to measure learning impact over time. The Streamlit tool has not been stress-tested for handling very large files, heavy concurrency, code switching, or noisy transcripts. Reproducibility depends on evolving third-party checkpoints and licensing that may limit the redistribution of results.

These constraints do not negate the findings, but they mark the current boundary of claims and point to clear next steps in data breadth, label depth, context modelling, calibration, and rigorous human-in-the-loop evaluation.

6.3 Recommendations

This project should strengthen evaluation practice before expanding functionality. Future work is needed to run a parallel benchmark on a well-known public dataset, so that the reported scores can be compared with established results. An independent annotation audit should also be conducted on a stratified sample with inter-annotator agreement and confidence intervals. A small, carefully curated subset with transparent guidelines would help others reproduce results without repeating the full corpus effort.

Next, the modelling roadmap should prioritise a native fallacy detector trained on debate text. A staged plan would make sense, beginning with weak supervision from existing fallacy resources and moving to limited expert review on the most uncertain cases. Joint or multi-task learning that shares a backbone for argument typing and fallacy cues should be explored, together with short context windows across adjacent sentences and speaker turns. Experiments with light graph structure over sentence nodes could help the system reason about local links without heavy computation. Calibration should be added so that confidence estimates are meaningful, with temperature scaling, class-wise thresholds, and abstention options for uncertain predictions.

Deployment should focus on speed, footprint, and robustness. Knowledge distillation from the strongest transformer to a smaller student model would keep performance while reducing size and latency. Mixed-precision inference, quantisation, and simple caching would further improve responsiveness on modest hardware. The interface should include a privacy-first offline mode, clear export options for educators, and a small telemetry framework that collects only aggregate, anonymous usage for iterative improvement.

Lastly, educational impact should be measured directly. A classroom pilot with educators should test whether the feedback improves revision quality and student confidence. The tool should offer rubric-aligned views, per-class exemplars drawn from high-quality segments, and a slider that switches between brief and detailed explanations. Error-analysis dashboards should help judges see common misclassifications and tailor instruction.

Overall, these steps would transform the current prototype into a reliable assistant that supports Sustainable Development Goal 4 by providing frequent, fair, and explainable feedback in real learning environments.

References

Abd-Eldayem, R. M. A. (2023). The relationship between cognitive bias and logical fallacies in Egyptian society. *Social Sciences*. https://doi.org/10.11648/j.ss.20231206.14

Adak, A. (2024, December 10). 💻 *Visual Studio Code (VS Code): a powerful code editor* 🚀. DEV Community. https://dev.to/aniruddhaadak/visual-studio-code-vs-code-a-powerful-code-editor-417e

Aebissa, M. (2023). Effective communication: the key to success. *Journal of Organizational Culture Communications and Conflict*, *27*(5), 1–2. https://www.abacademies.org/articles/Effective%20Communication:%20The%20Key-1939-4691-27-5-124.pdf

Alhindi, T., Chakrabarty, T., Musi, E., & Muresan, S. (2022). Multitask instruction-based prompting for fallacy recognition. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. https://doi.org/10.18653/v1/2022.emnlp-main.560

Amaral, F. C. D., Pinto, H. S., & Martins, B. (2023). Argumentation Mining from Textual Documents Combining Deep Learning and Reasoning. In *Lecture notes in computer science* (pp. 389–401). https://doi.org/10.1007/978-3-031-49008-8_31

Andres. (2024, November 12). *Streamlit vs Flask vs Django comparison - November 2024*. https://www.restack.io/docs/streamlit-knowledge-streamlit-vs-flask-vs-django

Archie, L. C. (n.d.). *Ad populum: Appeal to popularity*. https://philosophy.lander.edu/logic/popular.html

Azevedo, A. I. R. L., & Santos, M. F. (2008). KDD, SEMMA AND CRISP-DM: a PARALLEL OVERVIEW. *Repositório Científico Do Instituto Politécnico Do Porto*, 182–185. https://recipp.ipp.pt/bitstream/10400.22/136/3/KDD-CRISP-SEMMA.pdf

Baccini, E., & Hartmann, S. (2022). The myside Bias in Argument Evaluation: A Bayesian model. *Proceedings of the Annual Meeting of the Cognitive Science Society*, *44*(44). https://escholarship.org/content/qt8nq023zs/qt8nq023zs_noSplash_944c2b8ba5ca05bb8ef709bc02e1d089.pdf?t=recl59

Bao, J., Jin, B., Sun, Y., Zhang, Y., He, Y., & Xu, R. (2024). A Comparison-Based Framework for argument quality assessment. *Electronics*, *13*(20), 4088. https://doi.org/10.3390/electronics13204088

Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the dangers of stochastic parrots. *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, 610–623. https://doi.org/10.1145/3442188.3445922

Berthet, V. (2021). The Measurement of Individual differences in Cognitive Biases: a review and improvement. *Frontiers in Psychology*, *12*. https://doi.org/10.3389/fpsyg.2021.630177

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., . . . Amodei, D. (2020). Language Models are Few-Shot Learners. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2005.14165

Burke, J. (2023, September 25). *Why and how to use Google Colab*. Search Enterprise AI. https://www.techtarget.com/searchEnterpriseAI/tutorial/Why-and-how-to-use-Google-Colab

Cabessa, J., Hernault, H., & Mushtaq, U. (2024). In-Context Learning and Fine-Tuning GPT for argument mining. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2406.06699

Chen, G., Cheng, L., Luu, A. T., & Bing, L. (2024). Exploring the potential of large language models in computational argumentation. *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, *1*, 2309–2330. https://doi.org/10.18653/v1/2024.acl-long.126

Chen, G., Cheng, L., Tuan, L. A., & Bing, L. (2023). Exploring the potential of large language models in computational argumentation. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2311.09022

Chen, G. H., Chen, S., Liu, Z., Jiang, F., & Wang, B. (2024). Humans or LLMs as the Judge? A study on Judgement Bias. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 8301–8327. https://doi.org/10.18653/v1/2024.emnlp-main.474

Cholifah, A. N., Alfaruqy, D., Pustika, R., & Sunarsih, S. (2024). Unmasking logical fallacies: An analysis of the 2024 Indonesian Vice-Presidential debates. *Journal of Pragmatics and Discourse Research*, *4*(2), 159–167. https://doi.org/10.51817/jpdr.v4i2.961

Code snippet: create Components without any frontend tooling (no React, Babel, Webpack, etc). (2021, May 18). Streamlit. https://discuss.streamlit.io/t/code-snippet-create-components-without-any-frontend-tooling-no-react-babel-webpack-etc/13064

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297. https://doi.org/10.1007/bf00994018

Corvaglia, L. (2024). *Strategies for critical thinking and resilience to manipulative messaging*. https://www.researchgate.net/publication/385284833_Strategies_for_Critical_Thinking_and_Resilience_to_Manipulative_Messaging

DeBERTa. (n.d.). https://huggingface.co/docs/transformers/v4.56.0/en/model_doc/deberta

Deploy - Streamlit docs. (n.d.). https://docs.streamlit.io/deploy

Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). QLORA: Efficient Finetuning of Quantized LLMS. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2305.14314

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, *1*, 4171–4186. https://doi.org/10.18653/v1/n19-1423

Ding, N., Qin, Y., Yang, G., Wei, F., Yang, Z., Su, Y., Hu, S., Chen, Y., Chan, C., Chen, W., Yi, J., Zhao, W., Wang, X., Liu, Z., Zheng, H., Chen, J., Liu, Y., Tang, J., Li, J., & Sun, M. (2023). Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, *5*(3), 220–235. https://doi.org/10.1038/s42256-023-00626-4

Dutta, S., Juneja, J., Das, D., & Chakraborty, T. (2022). Can Unsupervised Knowledge Transfer from Social Discussions Help Argument Mining? *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. https://doi.org/10.18653/v1/2022.acl-long.536

Fadilpašić, S. (2025, March 19). *Best IDE for Python of 2025*. TechRadar. https://www.techradar.com/best/best-ide-for-python

Favero, L., Pérez-Ortiz, J. A., Käser, T., & Oliver, N. (2025). Leveraging small LLMs for argument Mining in Education: argument Component identification, Classification, and assessment. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2502.14389

Fine-Tuning DeepSeek R1 (Reasoning Model). (2025, January 27). Datacamp. https://www.datacamp.com/tutorial/fine-tuning-deepseek-r1-reasoning-model

Freeman, R. (n.d.). Chapter 3 Fallacies. In *Logical Reasoning*. https://logicalreasoning.net/CHAPTER3.pdf

Friedman, H. H. (2023). Cognitive biases that interfere with critical thinking and scientific reasoning: a course module. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.2958800

Friedman, H. H. (2024). Logical fallacies: how they undermine critical thinking and how to avoid them. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.4794200

Gallegos, I. O., Rossi, R. A., Barrow, J., Tanjim, M. M., Kim, S., Dernoncourt, F., Yu, T., Zhang, R., & Ahmed, N. K. (2023). Bias and Fairness in Large Language Models: A survey. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2309.00770

Gao, T., Jin, J., Ke, Z. T., & Moryoussef, G. (2025). A comparison of DeepSeek and other LLMs. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2502.03688

GeeksforGeeks. (2024, November 15). *Best Python libraries for Machine Learning*. GeeksforGeeks. https://www.geeksforgeeks.org/best-python-libraries-for-machine-learning/

GeeksforGeeks. (2025a, January 2). *History of Python*. GeeksforGeeks. https://www.geeksforgeeks.org/history-of-python/

GeeksforGeeks. (2025b, January 28). *KDD process in databases*. GeeksforGeeks. https://www.geeksforgeeks.org/kdd-process-in-data-mining/

GeeksforGeeks. (2025c, April 4). *What is Python? Its Uses and Applications*. GeeksforGeeks. https://www.geeksforgeeks.org/what-is-python/

GeeksforGeeks. (2025d, July 18). *Optuna*. GeeksforGeeks. https://www.geeksforgeeks.org/machine-learning/optuna/

GeeksforGeeks. (2025e, July 23). *10 Best Python IDEs to Use [2025]*. GeeksforGeeks. https://www.geeksforgeeks.org/python/top-python-ide/

GeeksforGeeks. (2025f, July 23). *How SVM constructs boundaries?* GeeksforGeeks. https://www.geeksforgeeks.org/machine-learning/how-svm-constructs-boundaries/

GeeksforGeeks. (2025g, August 13). *Understanding TFIDF (Term FrequencyInverse Document Frequency)*. GeeksforGeeks. https://www.geeksforgeeks.org/machine-learning/understanding-tf-idf-term-frequency-inverse-document-frequency/

Ghahreman, N., & Dastjerdi, A. B. (2011). Semi-Automatic labeling of training data sets in text classification. *Computer and Information Science*, *4*(6). https://doi.org/10.5539/cis.v4n6p48

Gielens, E., & Sowula, J. (2024). *Goodbye Human Annotators? Content analysis of policy debates using ChatGPT*. https://doi.org/10.31235/osf.io/m5r3h

Goal 4: Quality Education - the global goals. (2024, January 23). The Global Goals. https://www.globalgoals.org/goals/4-quality-education/

Goffredo, P., Espinoza, M., Villata, S., & Cabrio, E. (2023). Argument-based detection and classification of fallacies in political debates. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 11101–11112. https://doi.org/10.18653/v1/2023.emnlp-main.684

Goodfellow, I., Bengio, Y., & Courville, A. (2018). *Deep learning*. MIT Press. https://www.deeplearningbook.org/contents/intro.html

Gorur, D., Rago, A., & Toni, F. (2024). Can Large Language Models perform Relation-based Argument Mining? *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2402.11243

Graham, M. (2025, January 11). *What is Replit? A beginner's guide | Rapid Dev*. RapidDev. https://www.rapidevelopers.com/blog/what-is-replit-a-beginners-guide

Gupta, A., Zuckerman, E., & O'Connor, B. (2024). Harnessing Toulmin's theory for zero-shot argument explication. *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, *1*, 10259–10276. https://doi.org/10.18653/v1/2024.acl-long.552

Hacking, C., Verbeek, H., Hamers, J. P. H., & Aarts, S. (2023). Comparing text mining and manual coding methods: Analysing interview data on quality of care in long-term care for older adults. *PLoS ONE*, *18*(11), e0292578. https://doi.org/10.1371/journal.pone.0292578

Haddadan, S., Cabrio, E., & Villata, S. (2019). Yes, we can! Mining arguments in 50 years of US presidential campaign debates. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 4684–4690. https://doi.org/10.18653/v1/p19-1463

Hammond, M., & Charalampidi, M. (2020). Should Britain leave the EU? An exploration of online argument through a Toulmin perspective. *Journal on Educational Technology*, *28*(1), 5–19. https://doi.org/10.17471/2499-4324/1097

Harly, W., & Girsang, A. S. (2022). CNN-BERT for measuring agreement between argument in online discussion. *International Journal of Web Information Systems*, *18*(5/6), 356–368. https://doi.org/10.1108/ijwis-12-2021-0141

Hasan, M. K., Spann, J., Hasan, M., Islam, M. S., Haut, K., Mihalcea, R., & Hoque, E. (2021). Hitting your MARQ: Multimodal ARgument Quality Assessment in Long Debate Video. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 6387–6397. https://doi.org/10.18653/v1/2021.emnlp-main.515

He, P., Liu, X., Gao, J., & Chen, W. (2020). DeBERTa: Decoding-enhanced BERT with Disentangled Attention. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2006.03654

Heinrichs, F. (2025, January 26). *Using CRISP-DM to grow as data scientist*. Towards Data Science. https://towardsdatascience.com/using-crisp-dm-to-grow-as-data-scientist-a07ce3fd9d56/

Heller, M. (2022, July 8). *What is Visual Studio Code? Microsoft's extensible code editor*. InfoWorld. https://www.infoworld.com/article/2335960/what-is-visual-studio-code-microsofts-extensible-code-editor.html

Holtermann, C., Lauscher, A., & Ponzetto, S. (2022). Fair and argumentative language modeling for computational argumentation. *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 7841–7861. https://doi.org/10.18653/v1/2022.acl-long.541

Hossain, M. (2024, December 11). *How Python's Rich Ecosystem of Libraries is Transforming the Way Developers Approach Complex Projects*. DEV Community. https://dev.to/marufhossain/how-pythons-rich-ecosystem-of-libraries-is-transforming-the-way-developers-approach-complex-534g

IBM. (2024, February 12). Transfer learning. *IBM*. https://www.ibm.com/think/topics/transfer-learning

Irani, A., Park, J. Y., Esterling, K., & Faloutsos, M. (2025). WIBA: What is being argued? A comprehensive approach to argument mining. In *Lecture notes in computer science* (pp. 337–354). https://doi.org/10.1007/978-3-031-78541-2_21

Jin, Z., Lalwani, A., Vaidhya, T., Shen, X., Ding, Y., Lyu, Z., Sachan, M., Mihalcea, R., & Schoelkopf, B. (2022). Logical fallacy detection. *Findings of the Association for Computational Linguistics: EMNLP 2022*, 7180–7198. https://doi.org/10.18653/v1/2022.findings-emnlp.532

Jupyter Notebooks in VS code. (2021, November 3). https://code.visualstudio.com/docs/datascience/jupyter-notebooks

Kanadan, M. (2024, November 23). *Roberta-large-fallacy-classification*. Hugging Face. https://huggingface.co/MidhunKanadan/roberta-large-fallacy-classification

Khoirunisa, A., & Indah, R. N. (2022). Argumentative Statements in the 2016 presidential debates of the U.S: A Critical Discourse analysis. *JEELS (Journal of English Education and Linguistics Studies)*, *4*(2), 155–173. https://doi.org/10.30762/jeels.v4i2.64

Kim, M. (2021). A data mining framework for financial prediction. *Expert Systems With Applications*, *173*, 114651. https://doi.org/10.1016/j.eswa.2021.114651

Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1408.5882

kredor/punctuate-all · Hugging Face. (2022, December 6). https://huggingface.co/kredor/punctuate-all

Krippendorff, K. (2011). *Computing Krippendorff's Alpha-Reliability*. https://www.asc.upenn.edu/usr/krippendorff/mwebreliability5.pdf

Lalwani, A., Chopra, L., Hahn, C., Trippel, C., Jin, Z., & Sachan, M. (2024). NL2FOL: Translating Natural Language to First-Order Logic for Logical Fallacy Detection. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2405.02318

Lauscher, A., Lüken, T., & Glavaš, G. (2021). Sustainable modular debiasing of language models. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2109.03646

Lawrence, J., & Reed, C. (2019). Argument Mining: a survey. *Computational Linguistics*, *45*(4), 765–818. https://doi.org/10.1162/coli_a_00364

Lei, Y., & Huang, R. (2024). Boosting logical fallacy reasoning in LLMs via Logical Structure tree. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2410.12048

Li, J. (2024). Moving beyond text: Multi-modal expansion of the Toulmin model for enhanced AI legal reasoning. In *Communications in computer and information science* (pp. 299–308). https://doi.org/10.1007/978-981-97-0065-3_23

Li, J., Liu, X., Zhang, K., Liu, Y., Tang, Z., Spirtes, P., & Leqi, L. (2024). Steering LLMs towards Unbiased Responses: A Causality-Guided Debiasing Framework. *ICLR 2024 Workshop on Secure and Trustworthy Large Language Models*. https://openreview.net/forum?id=RYdozB0GdB

Lippi, M., & Torroni, P. (2016). Argumentation mining. *ACM Transactions on Internet Technology*, *16*(2), 1–25. https://doi.org/10.1145/2850417

Long, L., Minervini, A., & Gladd, J. (n.d.). *Toulmin Argument model*. Pressbooks. https://pressbooks.calstate.edu/writingargumentsinstem/chapter/toulmin-argument-model/

Malaviarachchi, U. T. (2024, June 4). *Remote development: using VSCode for remote coding and collaboration*. DEV Community. https://dev.to/umeshtharukaofficial/remote-development-using-vscode-for-remote-coding-and-collaboration-36ep

Mambrol, N. (2016, March 17). *Intentional fallacy*. Literary Theory and Criticism. https://literariness.org/2016/03/17/intentional-fallacy/

Manning, C. D., Raghavan, P., & Schütze, H. (2009). Introduction to information retrieval. *Choice Reviews Online*, *46*(05), 46–2715. https://doi.org/10.5860/choice.46-2715

Mao, T., Fu, J., & Yoshie, O. (2024a). Unearthing the efficacy of ChatGPT in argumentation Analysis: Performance, Potentials and Limitations. *2024 4th Asia Conference on Information Engineering (ACIE)*, 157–162. https://doi.org/10.1109/acie61839.2024.00033

Mao, T., Fu, J., & Yoshie, O. (2024b). Enhancing argument pair extraction through supervised Fine-Tuning of the llama model. *2024 IEEE 3rd International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA)*, 1153–1158. https://doi.org/10.1109/eebda60612.2024.10485782

Mao, T., Yoshie, O., Fu, J., & Mao, W. (2023). Seeing both sides: context-aware heterogeneous graph matching networks for extracting-related arguments. *Neural Computing and Applications*, *36*(9), 4741–4762. https://doi.org/10.1007/s00521-023-09250-0

Marga, I. (2024, April 26). *Python VS other languages in AI Development | UNI*. https://www.uni.agency/post/python-vs-other-languages-the-dominance-in-ai-development

Matt. (2025, July 6). *Kaiming normal Initialization in PyTorch: A Comprehensive guide*. Codegenes. https://www.codegenes.net/blog/kaiming-normal-pytorch/

Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., & Galstyan, A. (2021). A survey on Bias and Fairness in Machine Learning. *ACM Computing Surveys*, *54*(6), 1–35. https://doi.org/10.1145/3457607

Mirzababaei, B., & Pammer-Schindler, V. (2021). Developing a conversational agent's capability to identify structural wrongness in arguments based on Toulmin's model of arguments. *Frontiers in Artificial Intelligence*, *4*. https://doi.org/10.3389/frai.2021.645516

Mushtaq, U., & Cabessa, J. (2023). Argument Mining with Modular BERT and Transfer Learning. *2022 International Joint Conference on Neural Networks (IJCNN)*, 1–8. https://doi.org/10.1109/ijcnn54540.2023.10191968

Nasir, J. (2025, February 9). Fine-Tuning DeepSeek R1 :A Step by Step Guide - Jamal Nasir - Medium. *Medium*. https://medium.com/@coppeliasim1122/fine-tuning-deepseek-r1-a-step-by-step-guide-7517ec9424ed

Nguyen, H., & Litman, D. (2018). Argument mining for improving the automated scoring of persuasive essays. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1). https://doi.org/10.1609/aaai.v32i1.12046

Niculae, V., Park, J., & Cardie, C. (2017). Argument Mining with Structured SVMs and RNNs. *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. https://doi.org/10.18653/v1/p17-1091

Nikolopoulou, K. (2023, May 30). *False Dilemma Fallacy | Examples & Definition*. Scribbr. https://www.scribbr.com/fallacies/false-dilemma-fallacy/

Nilsson, N. J. (1998). *Introduction to Machine Learning: An early draft of a proposed textbook*. https://ai.stanford.edu/~nilsson/MLBOOK.pdf

Nippold, M. A. (2023). Unlocking logical fallacies: a key to building critical thinking skills in adolescents. *Perspectives of the ASHA Special Interest Groups*, *9*(1), 1–13. https://doi.org/10.1044/2023_persp-23-00108

Northcutt, C., Jiang, L., & Chuang, I. (2021). Confident Learning: Estimating uncertainty in Dataset labels. *Journal of Artificial Intelligence Research*, *70*, 1373–1411. https://doi.org/10.1613/jair.1.12125

oliverguhr/fullstop-punctuation-multilang-large · Hugging Face. (2022, December 6). https://huggingface.co/oliverguhr/fullstop-punctuation-multilang-large

Palacios, H. J. G., Toledo, R. a. J., Pantoja, G. a. H., & Navarro, Á. a. M. (2017). A comparative between CRISP-DM and SEMMA through the construction of a MODIS repository for studies of land use and cover change. *Advances in Science Technology and Engineering Systems Journal*, *2*(3), 598–604. https://doi.org/10.25046/aj020376

Patel, T. (2023). Machine learning and applications in argumentation mining. In *Terra Science and Education*. https://doi.org/10.36838/v6i1.28

Peldszus, A., & Stede, M. (2015). Joint prediction in MST-style discourse parsing for argumentation mining. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 938–948. https://doi.org/10.18653/v1/d15-1110

Petric, D. (2020). *Logical fallacies*. https://www.researchgate.net/publication/339288684_Logical_Fallacies

Pietron, M., Olszowski, R., & Gomułka, J. (2024). Efficient argument classification with compact language models and ChatGPT-4 refinements. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2403.15473

Pushkar, A. (2025, July 9). *What is PyCharm?* Intellipaat. https://intellipaat.com/blog/what-is-pycharm/

Qamar, R., & Zardari, B. A. (2023). Artificial Neural Networks: An Overview. *Mesopotamian Journal of Computer Science*, 130–139. https://doi.org/10.58496/mjcsc/2023/015

Qureshi, R., Es-Sebbani, N., Galárraga, L., Graham, Y., Couceiro, M., & Bouraoui, Z. (2024). REFINE-LM: Mitigating Language Model Stereotypes via Reinforcement Learning. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2408.09489

Ray, P. P. (2023). ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope. *Internet of Things and Cyber-Physical Systems*, *3*, 121–154. https://doi.org/10.1016/j.iotcps.2023.04.003

Reisert, P., Inoue, N., Okazaki, N., & Inui, K. (2015). A computational approach for generating Toulmin model argumentation. *Proceedings of the 2nd Workshop on Argumentation Mining*, 45–55. https://doi.org/10.3115/v1/w15-0507

Rescala, P., Ribeiro, M. H., Hu, T., & West, R. (2024). Can language models recognize convincing arguments? *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2404.00750

Rhomrasi, L., Ahsini, Y., Igualde-Sáez, A., Vinuesa, R., Hoyas, S., García-Sabater, J. P., Fullana-I-Alfonso, M. J., & Conejero, J. A. (2025). LLM performance on mathematical reasoning in Catalan language. *Results in Engineering*, 104366. https://doi.org/10.1016/j.rineng.2025.104366

Robbani, I., Reisert, P., Inoue, N., Pothong, S., Guerraoui, C., Wang, W., Naito, S., Choi, J., & Inui, K. (2024). Flee the flaw: annotating the underlying logic of fallacious arguments through templates and slot-filling. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2406.12402

Roush, A., & Balaji, A. (2020). DebateSum: A large-scale argument mining and summarization dataset. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2011.07251

Ruiz-Dolz, R., Heras, S., & Garcia, A. (2023). Automatic Debate Evaluation with Argumentation Semantics and Natural Language Argument Graph Networks. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 6030–6040. https://doi.org/10.18653/v1/2023.emnlp-main.368

Ruiz-Dolz, R., Nofre, M., Taulé, M., Heras, S., & García-Fornes, A. (2021). VivesDebate: a new annotated multilingual corpus of argumentation in a debate tournament. *Applied Sciences*, *11*(15), 7160. https://doi.org/10.3390/app11157160

Sahitaj, P., Ruiz-Dolz, R., Sahitaj, A., Nizamoglu, A., Schmitt, V., Mohtaj, S., & Möller, S. (2024). From Construction to Application: Advancing Argument Mining with the Large-Scale KIALOPRIME Dataset. In *Frontiers in artificial intelligence and applications*. https://doi.org/10.3233/faia240324

Sandu, B. (2025, August 29). *What is VSCode? Essential Features Uncovered*. TMS Outsource. https://tms-outsource.com/blog/posts/what-is-vscode/

SAS Help Center. (n.d.-a). https://documentation.sas.com/doc/en/emref/14.3/n061bzurmej4j3n1jnj8bbjjm1a2.htm

SAS Help Center. (n.d.-b). https://documentation.sas.com/doc/en/emref/14.3/n061bzurmej4j3n1jnj8bbjjm1a2.htm

Satishkumar, S. (2024). Toulmin's model of argumentative writing. In *San José State University Writing Center*. https://www.sjsu.edu/writingcenter/docs/handouts/Toulmin%20Model%20of%20Argumentative%20Writing.pdf

Schumann, J. (2019). *The Straw Man Fallacy. An experimental approach*. https://www.researchgate.net/publication/342211169_The_straw_man_fallacy_An_experimental_approach

Sen, S. (2025, January 28). DeepSeek: A journey of open source innovation in large language models. *Medium*. https://medium.com/@sudeshnasen/deepseek-a-journey-of-open-source-innovation-in-large-language-models-ab4920c99fdb

Shah, S., Patil, V., & Makwana, A. (2025). DeepSeek redefining AI excellence beyond OpenAI. *International Journal of Research Publication and Reviews*, *6*(2), 2581–2589. https://ijrpr.com/uploads/V6ISSUE2/IJRPR38836.pdf

Shargunam, S., Verma, A. K., Sourabh, R., Kumar, A., & Krishna, N. V. (2024). Stance detection and argument mining using BERT model. *Engineering Applications of Artificial Intelligence*. https://www.researchgate.net/publication/385505156_STANCE_DETECTION_AND_ARGUMENT_MINING_USING_BERT_MODEL

Singh, N. R. B. (2025). A Review Article for Argumentation Mining of Text through Machine Learning Techniques and Strategies. *Journal of Information Systems Engineering & Management*, *10*(26s), 326–345. https://doi.org/10.52783/jisem.v10i26s.4237

Smart Vision Europe. (2020, June 17). *Crisp DM methodology - Smart Vision Europe*. https://www.sv-europe.com/crisp-dm-methodology/

Sourati, Z., Venkatesh, V. P. P., Deshpande, D., Rawlani, H., Ilievski, F., Sandlin, H., & Mermoud, A. (2023). Robust and explainable identification of logical fallacies in natural language arguments. *Knowledge-Based Systems*, *266*, 110418. https://doi.org/10.1016/j.knosys.2023.110418

Stab, C., & Gurevych, I. (2017). Parsing argumentation structures in persuasive essays. *Computational Linguistics*, *43*(3), 619–659. https://doi.org/10.1162/coli_a_00295

Streamlit • A faster way to build and share data apps. (n.d.). https://streamlit.io/

Supriyadi, A. (2023). Improving students' critical thinking by employing Toulmin pattern of argumentation. *Education of English as a Foreign Language*, *6*(2), 130–144. https://doi.org/10.21776/ub.educafl.2023.006.02.03

Terra, J. (2024, May 21). What is transfer learning in machine learning? - Caltech. *Caltech* -. https://pg-p.ctme.caltech.edu/blog/ai-ml/what-is-transfer-learning-in-machine-learning

The Appeal to Emotion Fallacy: Arguing Through Feelings Rather than Facts. (n.d.). https://effectiviology.com/appeal-to-emotion

Thompson, S. (n.d.). *6 Part A: Stratified Sampling – Sampling theory and methods*. STAT 506. https://online.stat.psu.edu/stat506/Lesson06

Toledo-Ronen, O., Orbach, M., Bilu, Y., Spector, A., & Slonim, N. (2020). Multilingual Argument Mining: Datasets and Analysis. *Findings of the Association for Computational Linguistics: EMNLP 2020*, 303–317. https://doi.org/10.18653/v1/2020.findings-emnlp.29

Toulmin argument. (2023). In *School of Professional Studies, Trinity Washington University*. https://www.blinn.edu/writing-centers/pdfs/Toulmin-Argument.pdf

Toulmin, S. E. (2003). *The uses of argument*. Cambridge University Press. https://archive.org/details/stephen-e.-toulmin-the-uses-of-argument-2003/mode/2up

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., & Lample, G. (2023). LLAMA: Open and Efficient Foundation Language Models. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2302.13971

Trivedi, A. (2024, May 20). *Top 7 Python Libraries for data Visualization*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2024/05/top-python-libraries-for-data-visualization

Using HuggingFace with Streamlit. (2024, July 24). Streamlit. https://discuss.streamlit.io/t/using-huggingface-with-streamlit/75662

Verheij, B. (2005). Evaluating arguments based on Toulmin's scheme. *Argumentation*, *19*(3), 347–371. https://doi.org/10.1007/s10503-005-4421-z

Villata, S. (2024). The long road to trustworthy natural language argumentation. In *Frontiers in artificial intelligence and applications*. https://doi.org/10.3233/faia240304

What is Streamlit: All Why's and How's Answered | UI Bakery Blog. (n.d.). https://uibakery.io/blog/what-is-streamlit

Wilame. (2019, January 22). *Why is removing stop words not always a good idea*. Medium. https://medium.com/@wila.me/why-is-removing-stop-words-not-always-a-good-idea-c8d35bd77214

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., Von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., . . . Rush, A. (2020). Transformers: State-of-the-Art natural Language processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45. https://doi.org/10.18653/v1/2020.emnlp-demos.6

Yang, J., Latif, E., He, Y., & Zhai, X. (2025). Fine-Tuning ChatGPT for automatic scoring of written scientific explanations in Chinese. *Journal of Science Education and Technology*. https://doi.org/10.1007/s10956-025-10199-z

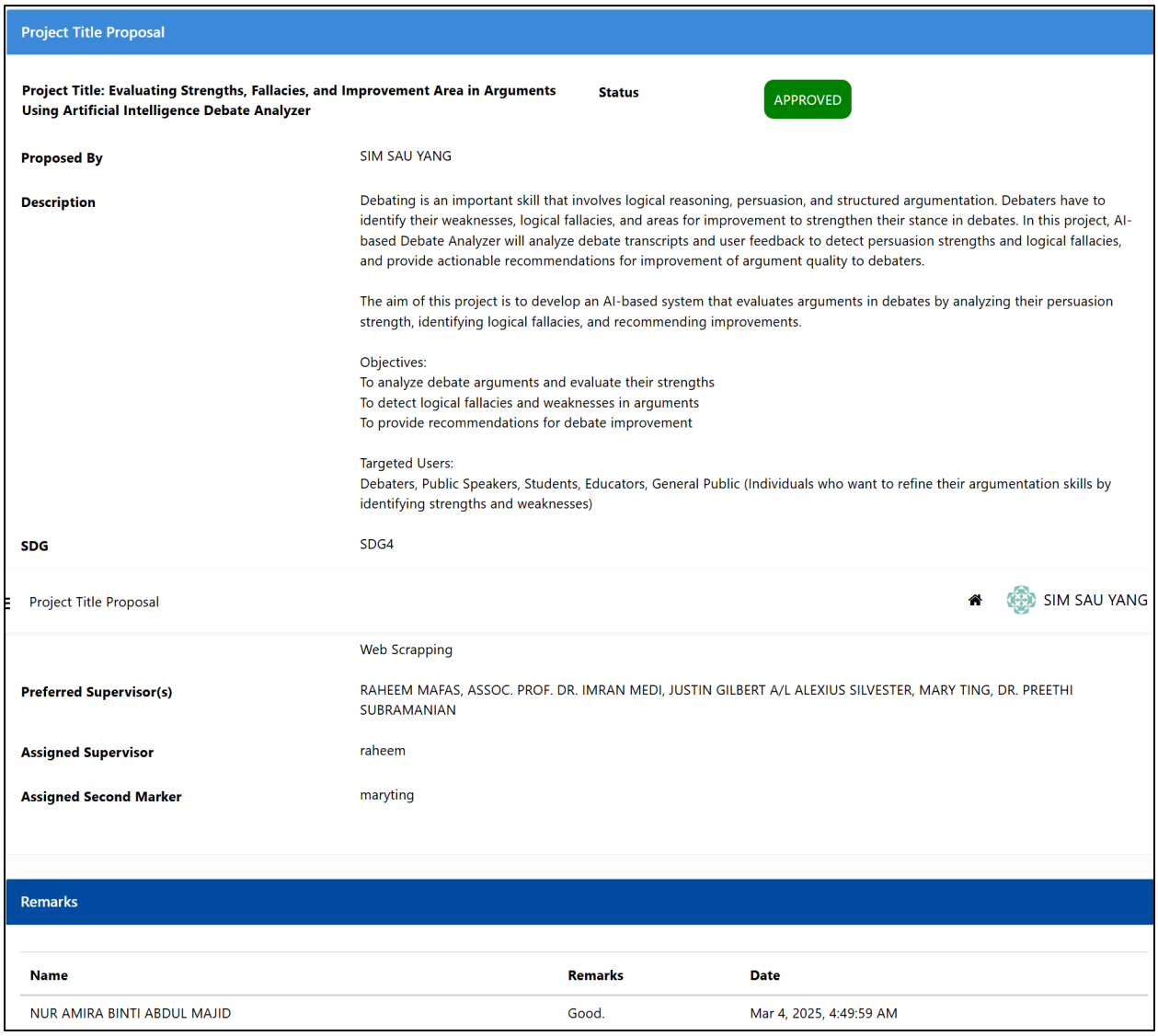
Zhang, G., Nulty, P., & Lillis, D. (2023). Argument mining with graph representation learning. *ICAIL '23: Proceedings of the Nineteenth International Conference on Artificial Intelligence and Law*, 371–380. https://doi.org/10.1145/3594536.3595152

Zhang, Y., & Sang, J. (2024). Inference-Time Rule Eraser: Distilling and removing bias rules to mitigate bias in deployed models. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2404.04814

Zhao, Z., Wallace, E., Feng, S., Klein, D., & Singh, S. (2021). Calibrate Before use: Improving Few-Shot performance of language models. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2102.09690

Appendices

Appendix A: PPF – Title Registration Proposal



Project Title Proposal

Project Title: Evaluating Strengths, Fallacies, and Improvement Area in Arguments Using Artificial Intelligence Debate Analyzer

Status APPROVED

Proposed By SIM SAU YANG

Description

Debating is an important skill that involves logical reasoning, persuasion, and structured argumentation. Debaters have to identify their weaknesses, logical fallacies, and areas for improvement to strengthen their stance in debates. In this project, AI-based Debate Analyzer will analyze debate transcripts and user feedback to detect persuasion strengths and logical fallacies, and provide actionable recommendations for improvement of argument quality to debaters.

The aim of this project is to develop an AI-based system that evaluates arguments in debates by analyzing their persuasion strength, identifying logical fallacies, and recommending improvements.

Objectives:
To analyze debate arguments and evaluate their strengths
To detect logical fallacies and weaknesses in arguments
To provide recommendations for debate improvement

Targeted Users:
Debaters, Public Speakers, Students, Educators, General Public (Individuals who want to refine their argumentation skills by identifying strengths and weaknesses)

SDG SDG4

Project Title Proposal SIM SAU YANG

Web Scrapping

Preferred Supervisor(s) RAHEEM MAFAS, ASSOC. PROF. DR. IMRAN MEDI, JUSTIN GILBERT A/L ALEXIUS SILVESTER, MARY TING, DR. PREETHI SUBRAMANIAN

Assigned Supervisor raheem

Assigned Second Marker maryting

Remarks

Name	Remarks	Date
NUR AMIRA BINTI ABDUL MAJID	Good.	Mar 4, 2025, 4:49:59 AM

Figure 123: Project Title Proposal

Appendix B: Ethics Forms (Fast Track)

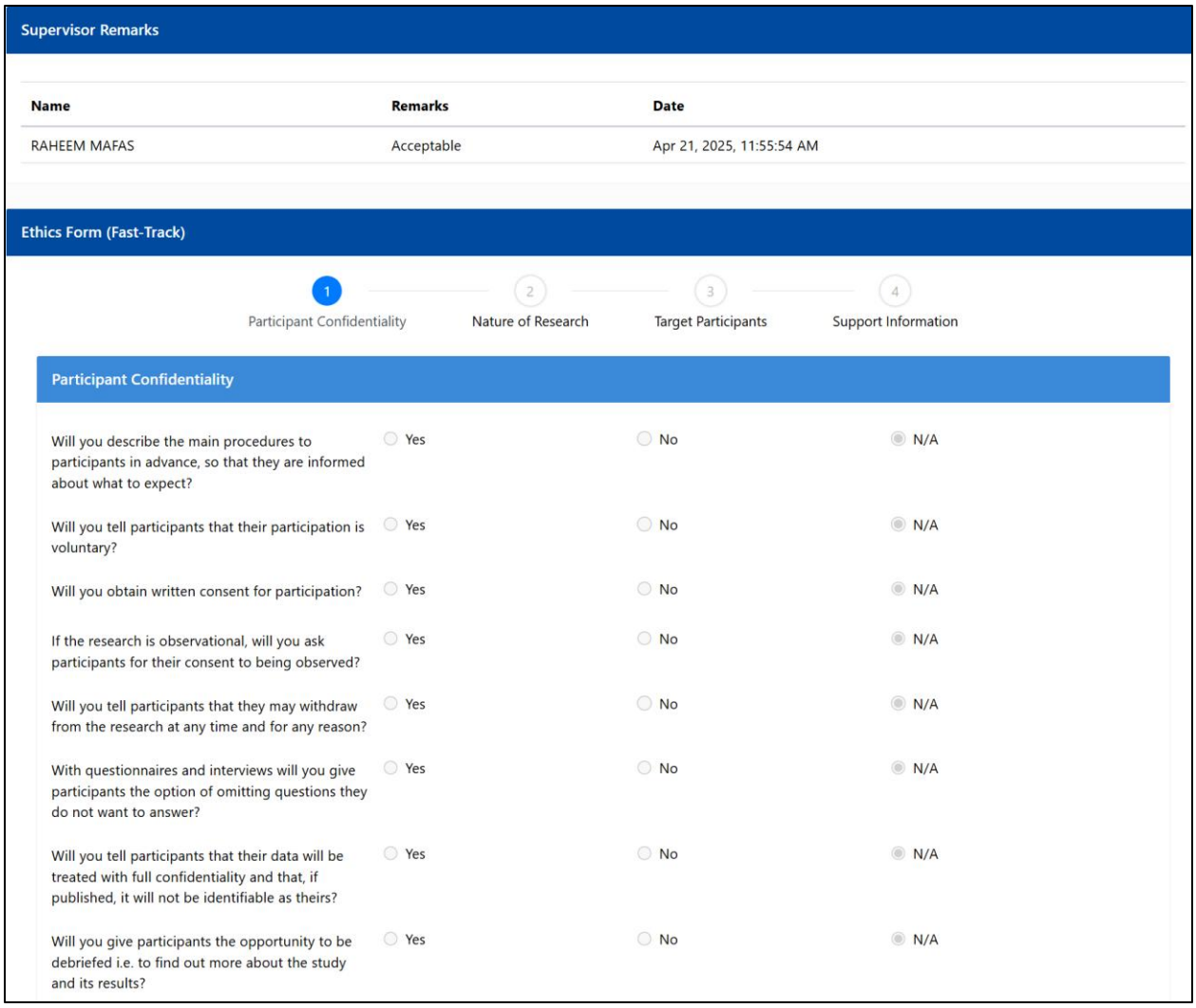


Figure 124: Ethnics Form

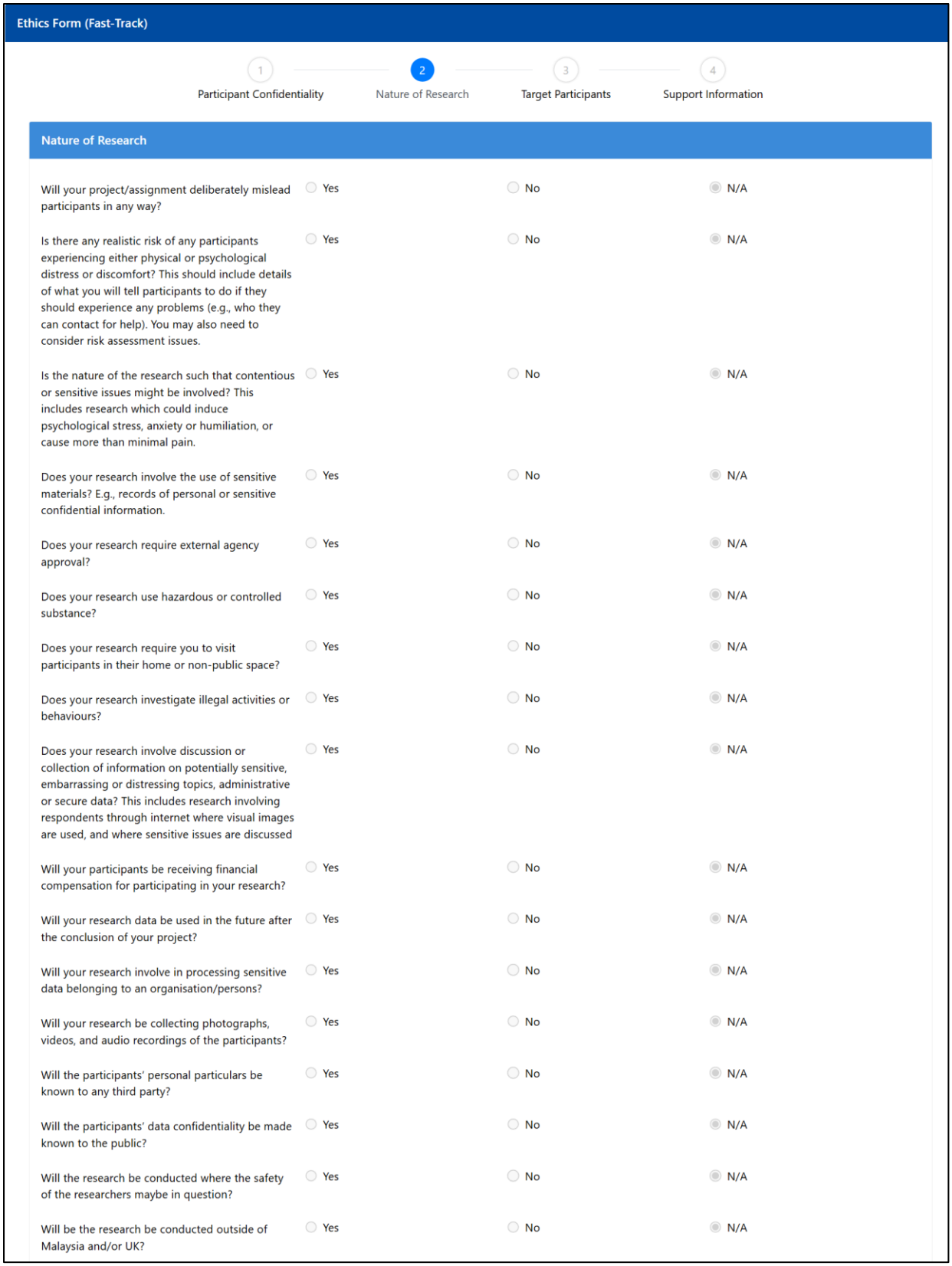


Figure 125: Ethnics Form

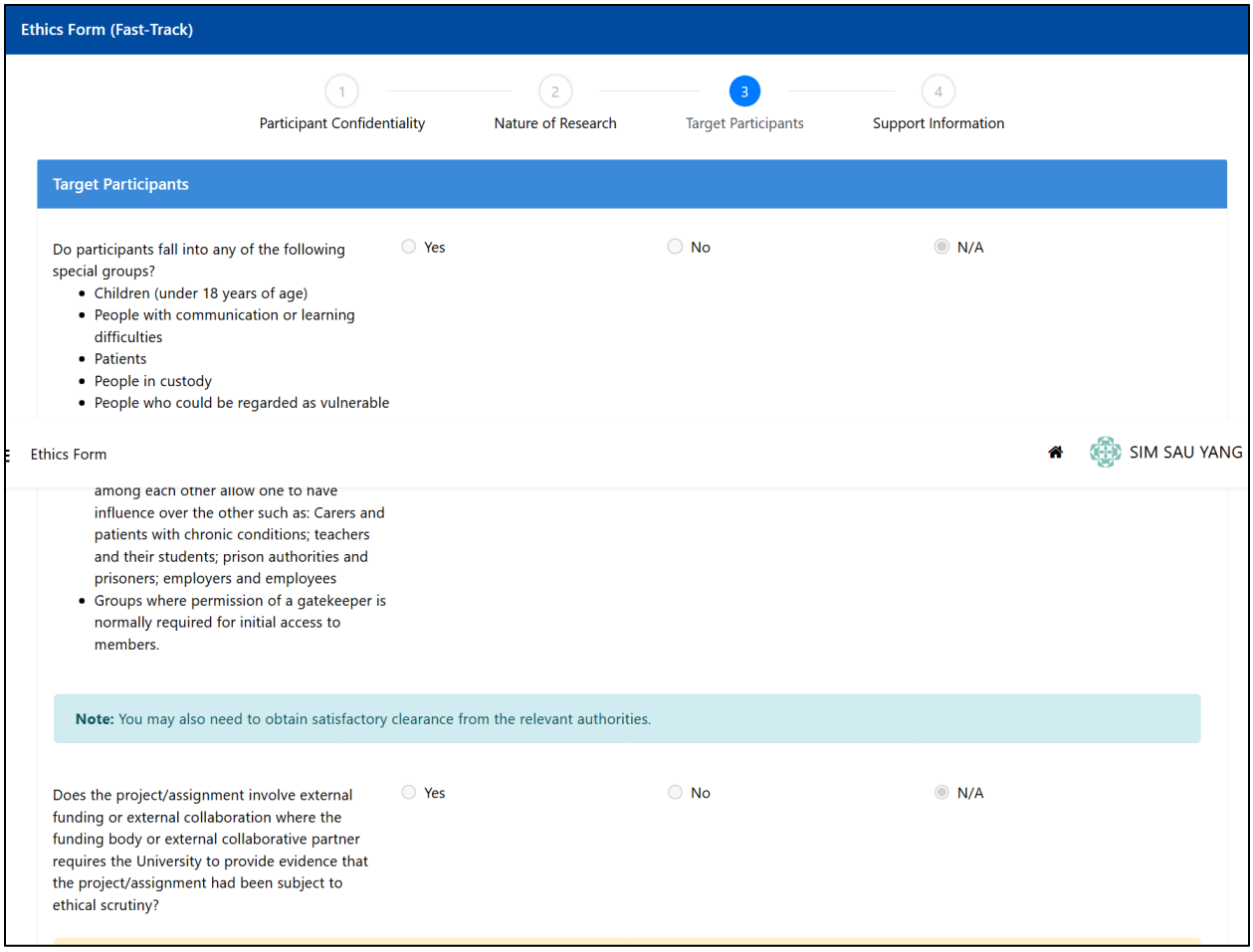


Figure 126: Ethnics Form

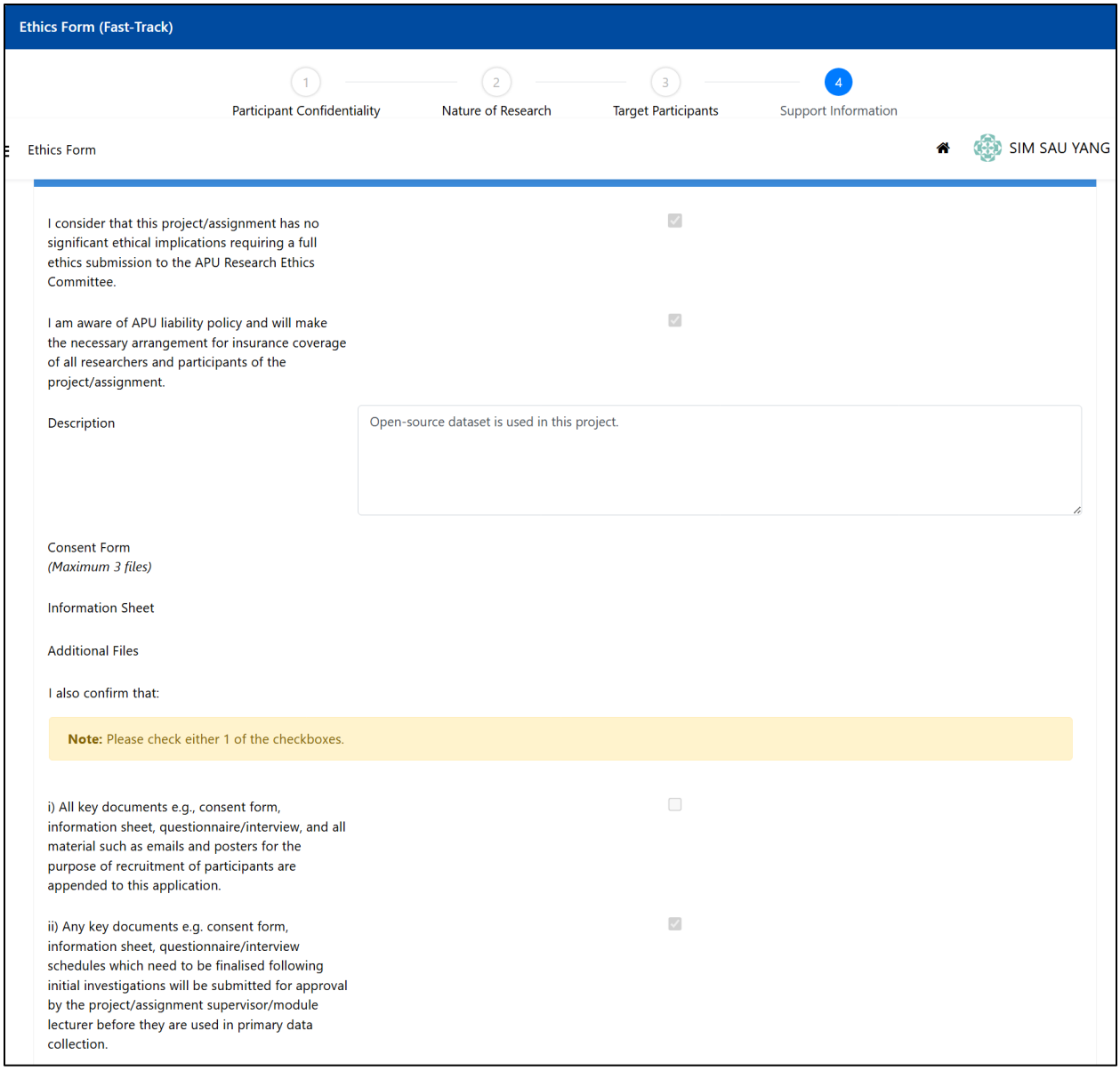
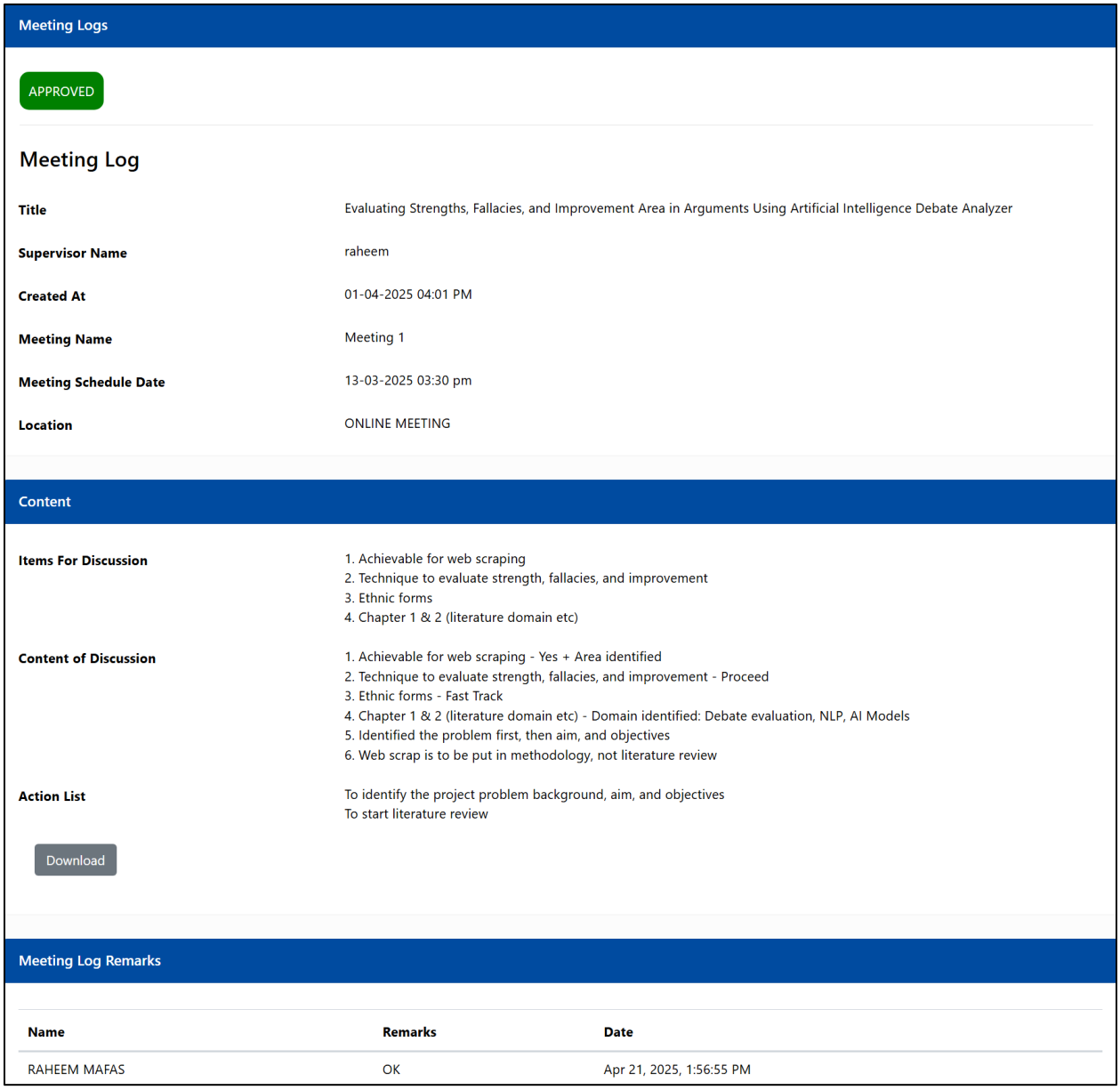


Figure 127: Ethnics Form

Appendix C: Meeting Log Sheets

Log Sheet 1



Meeting Logs

APPROVED

Meeting Log

Title	Evaluating Strengths, Fallacies, and Improvement Area in Arguments Using Artificial Intelligence Debate Analyzer
Supervisor Name	raheem
Created At	01-04-2025 04:01 PM
Meeting Name	Meeting 1
Meeting Schedule Date	13-03-2025 03:30 pm
Location	ONLINE MEETING

Content

Items For Discussion	1. Achievable for web scraping 2. Technique to evaluate strength, fallacies, and improvement 3. Ethnic forms 4. Chapter 1 & 2 (literature domain etc)
Content of Discussion	1. Achievable for web scraping - Yes + Area identified 2. Technique to evaluate strength, fallacies, and improvement - Proceed 3. Ethnic forms - Fast Track 4. Chapter 1 & 2 (literature domain etc) - Domain identified: Debate evaluation, NLP, AI Models 5. Identified the problem first, then aim, and objectives 6. Web scrap is to be put in methodology, not literature review
Action List	To identify the project problem background, aim, and objectives To start literature review

Download

Meeting Log Remarks

Name	Remarks	Date
RAHEEM MAFAS	OK	Apr 21, 2025, 1:56:55 PM

Figure 128: Log Sheet 1

Log Sheet 2

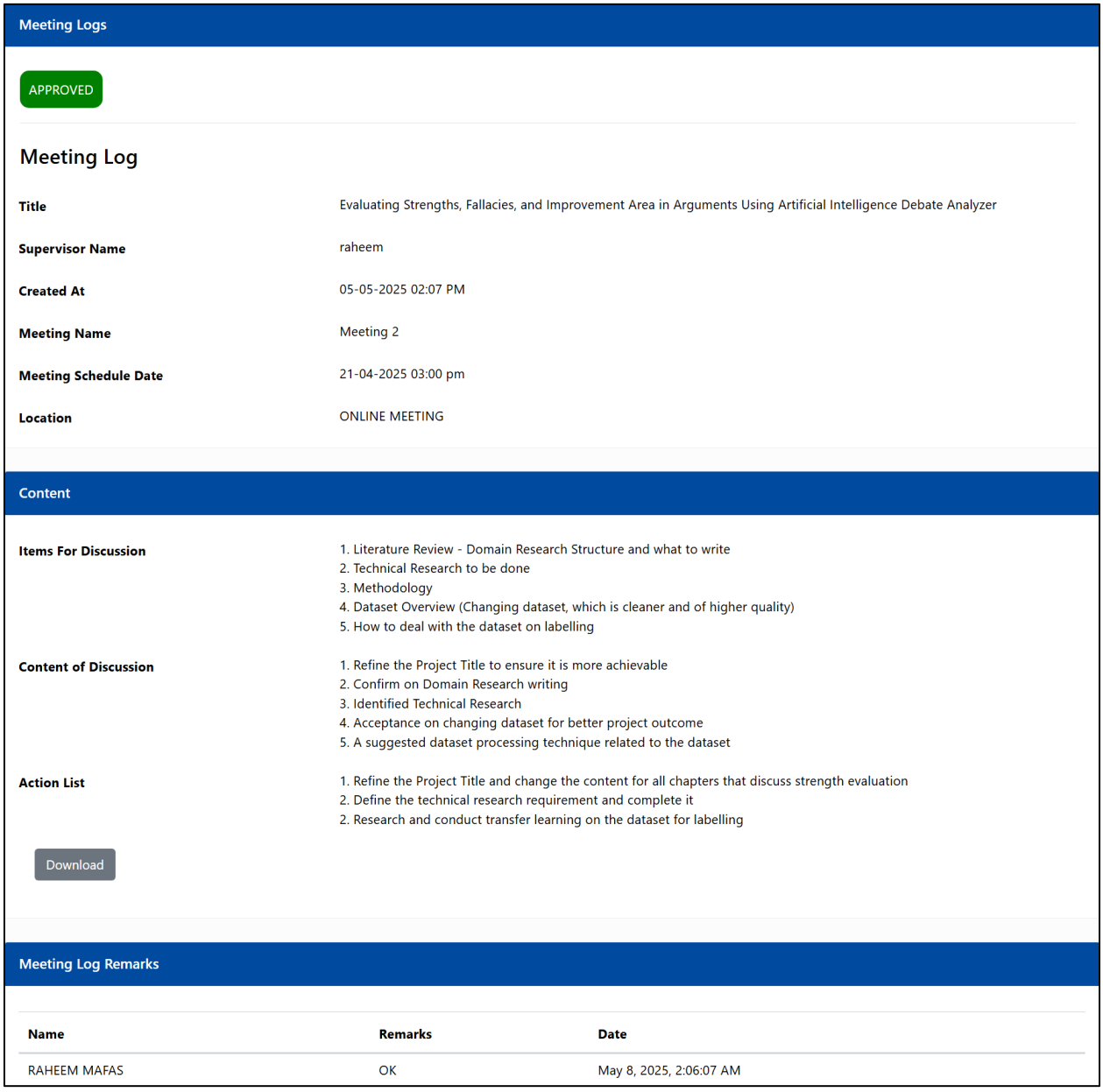


Figure 129: Log Sheet 2

Log Sheet 3

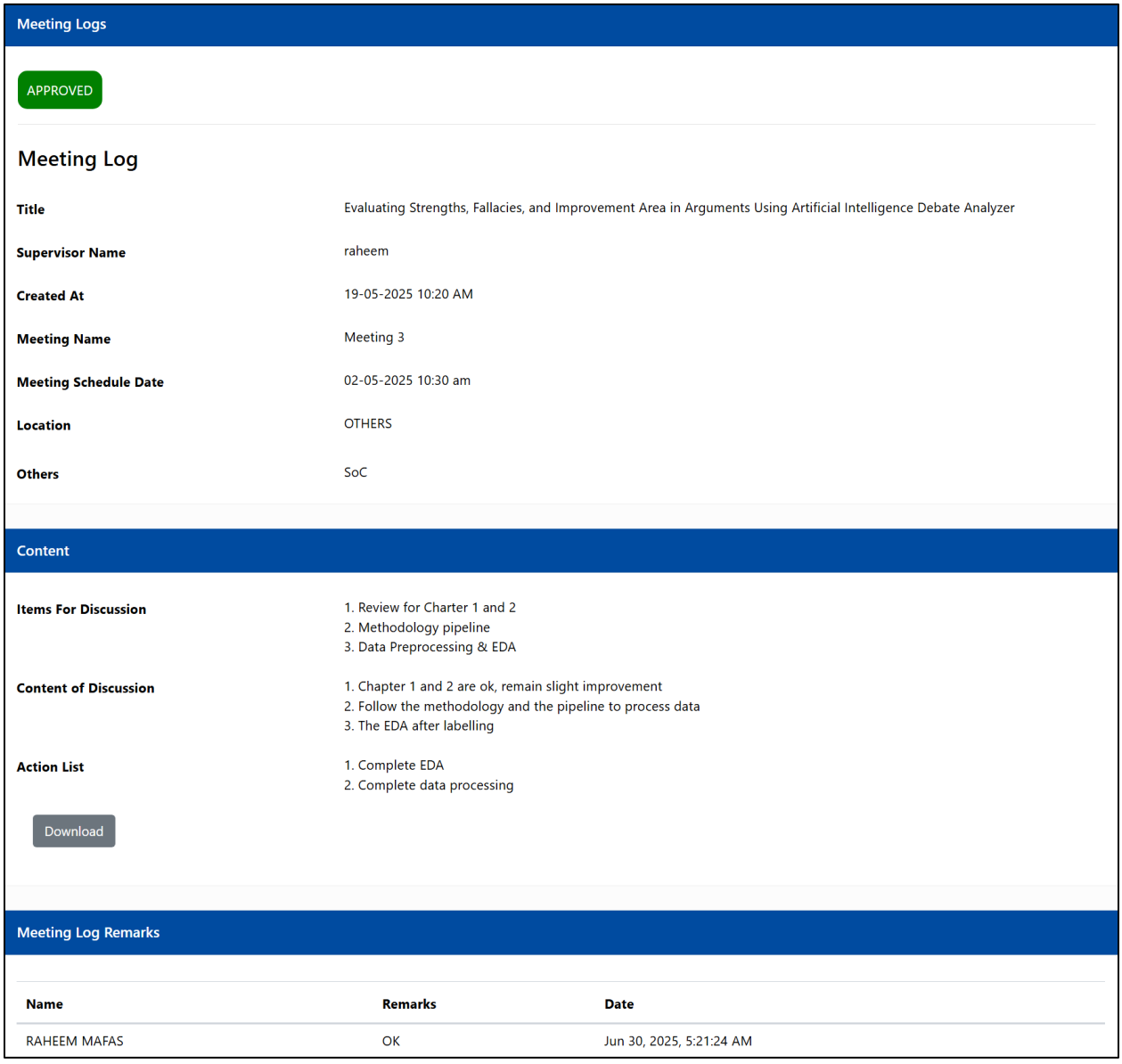


Figure 130: Log Sheet 3

Log Sheet 4

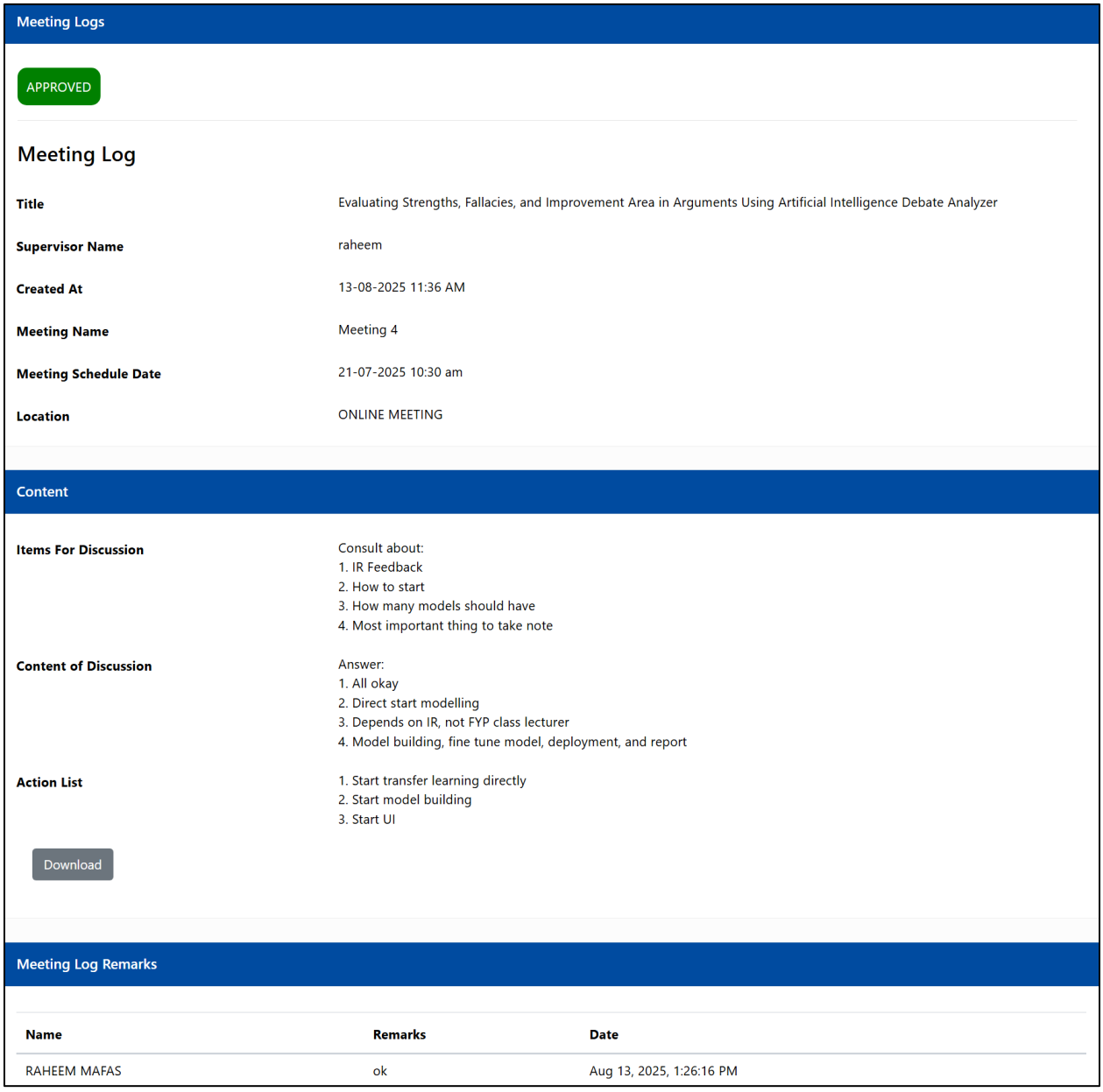


Figure 131: Log Sheet 4

Log Sheet 5

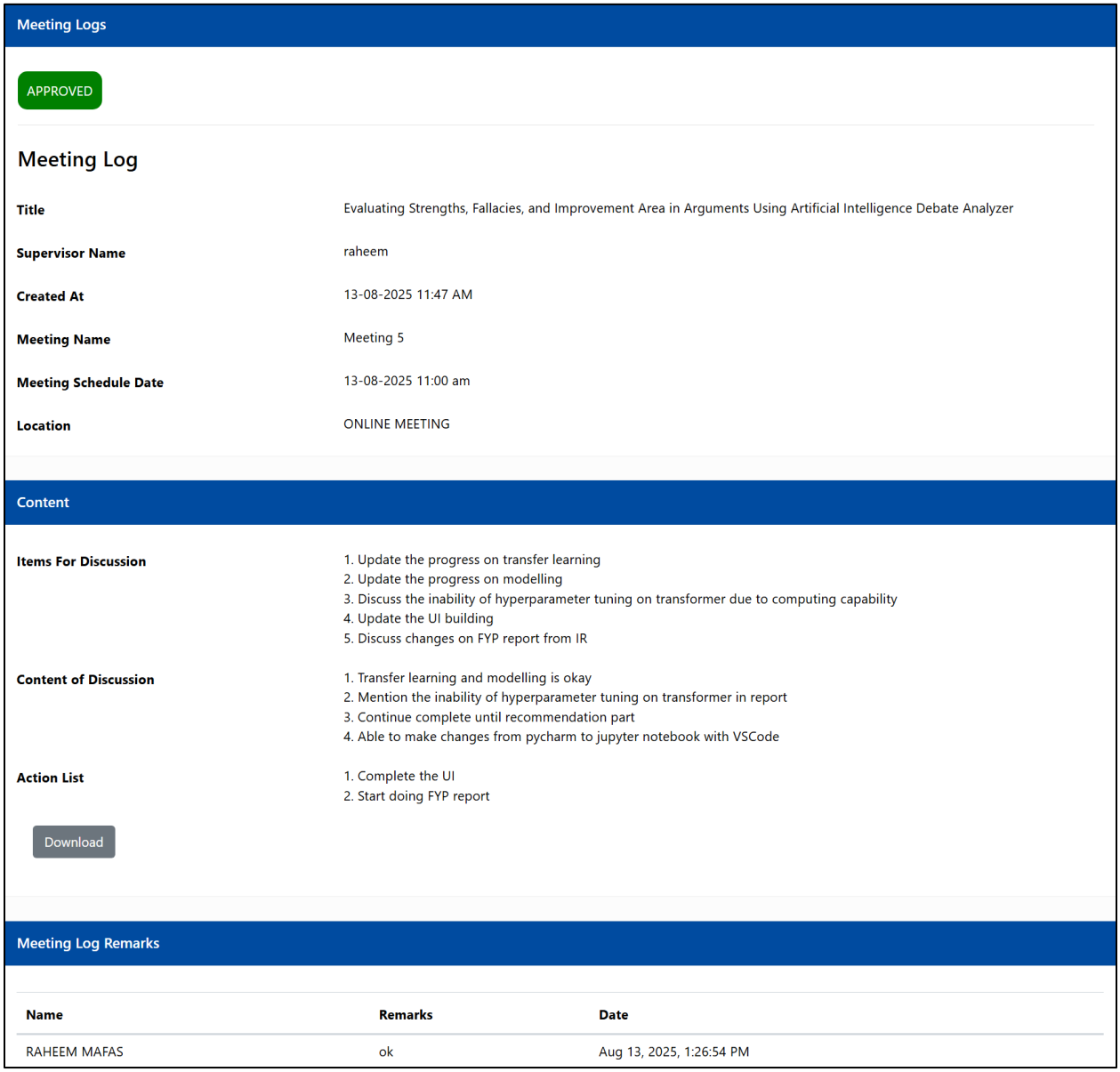


Figure 132: Log Sheet 5

Log Sheet 6

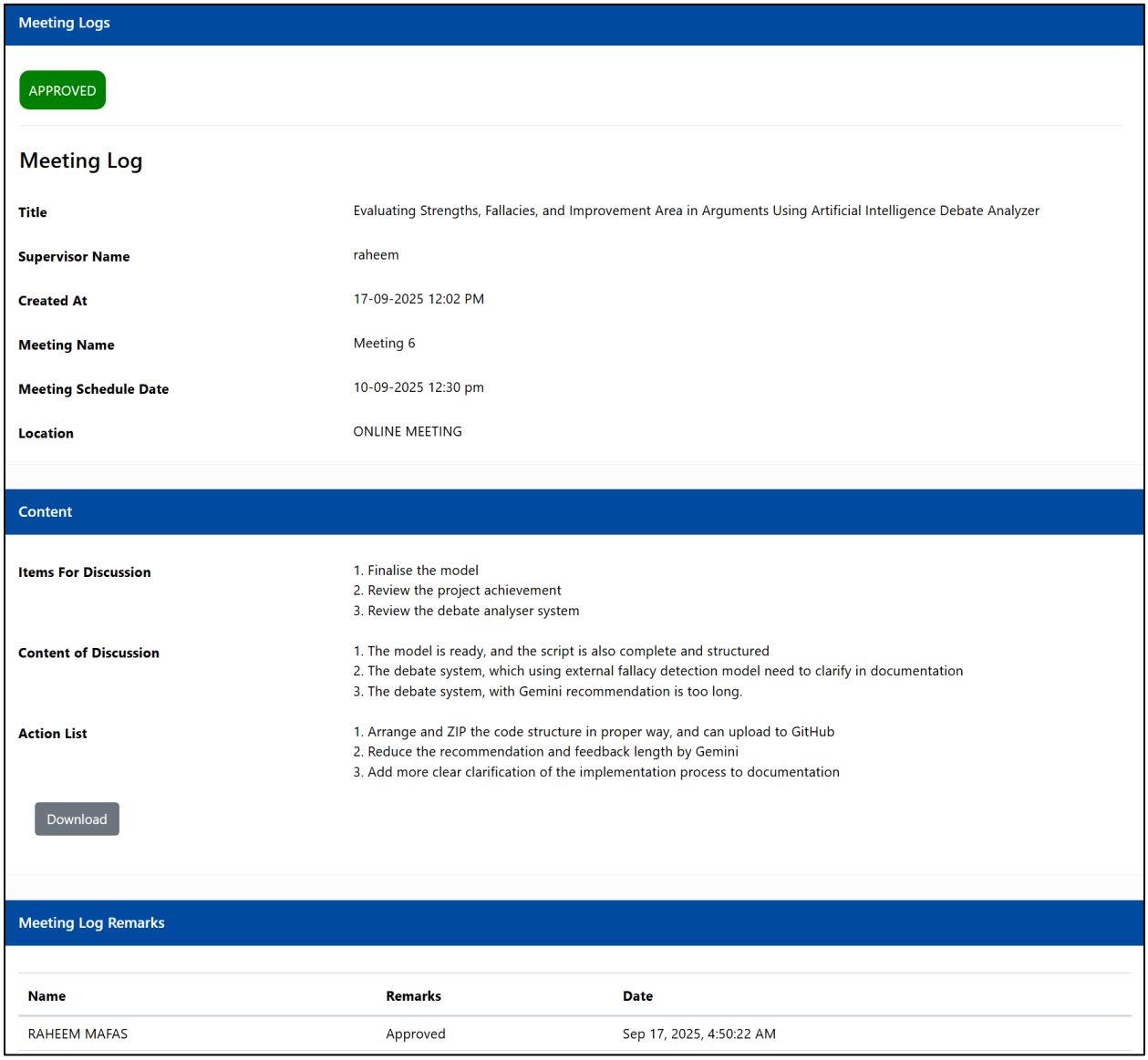
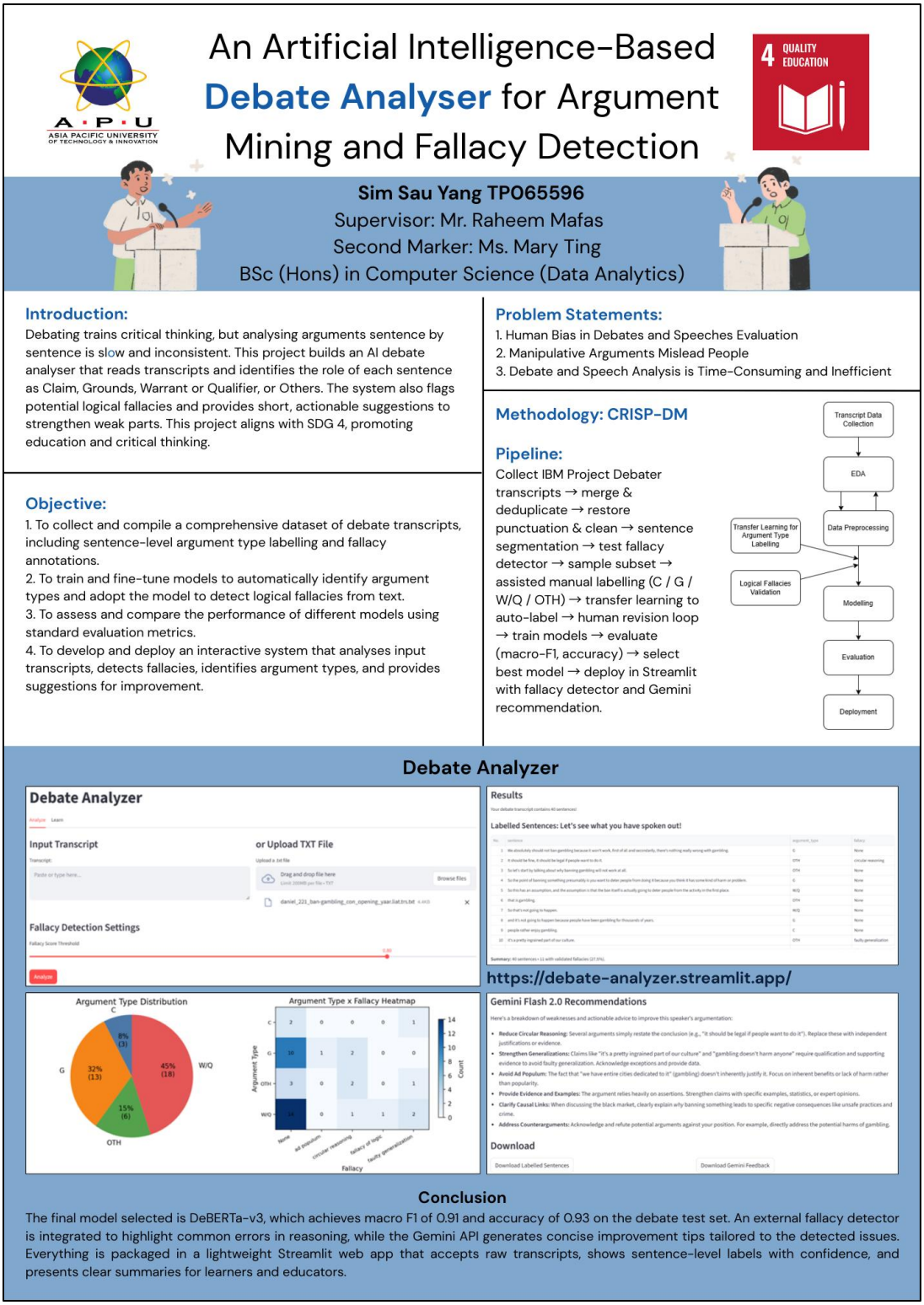


Figure 133: Log Sheet 6

Appendix D: Poster



Code Snippet 89: FYP Poster

Appendix E: Gantt Chart

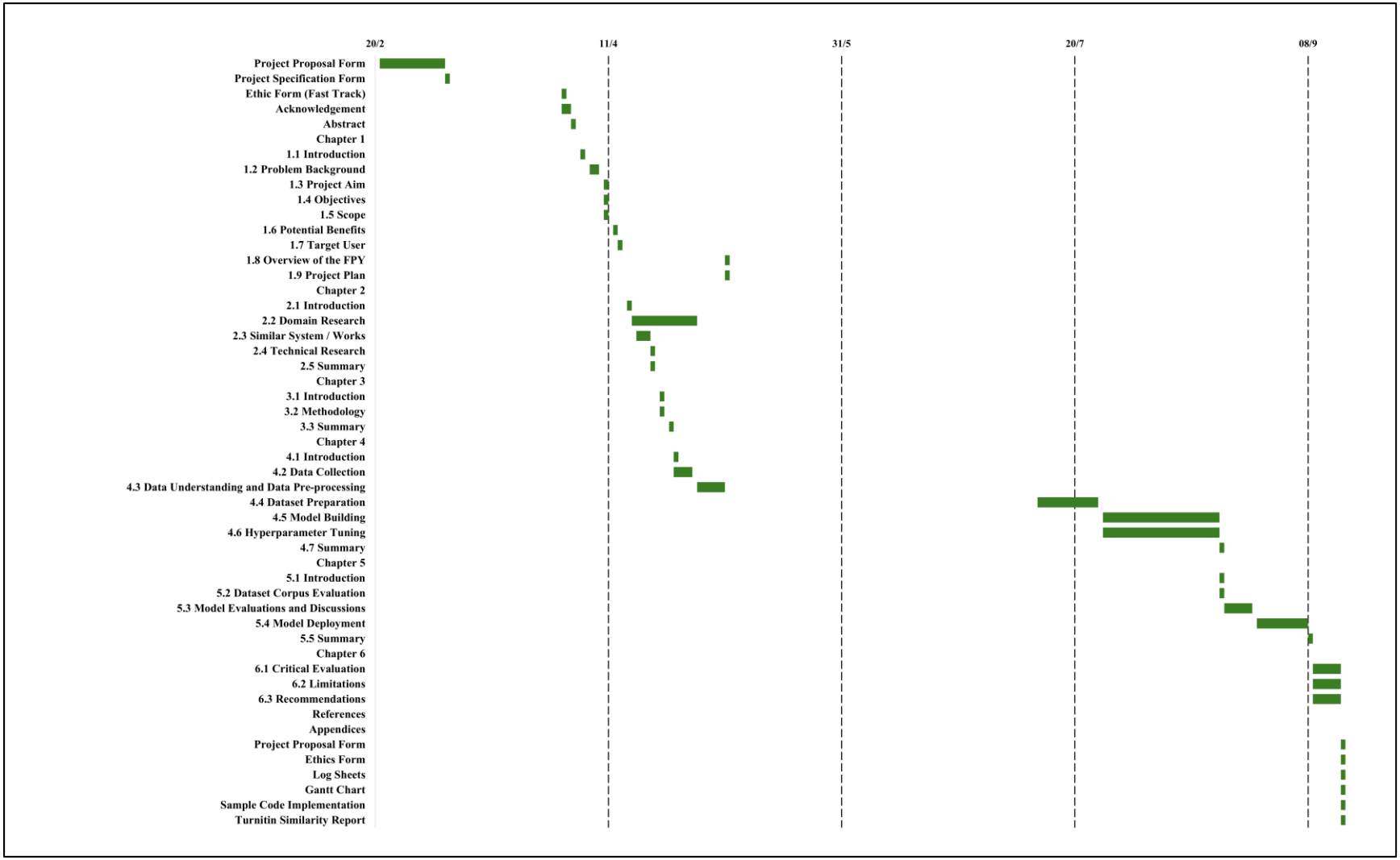


Figure 134: Gantt Chart

Appendix F: Sample Code Implementation

```
import os
os.environ["TRANSFORMERS_NO_TF"] = "1"  # Don't load TensorFlow
os.environ["TRANSFORMERS_NO_TORCHVISION"] = "1"  # Don't load Torchvision
os.environ["TF_ENABLE_ONEDNN_OPTS"] = "0"  # Disable TensorFlow CPU warnings

import streamlit as st
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import nltk
import pandas as pd
from io import StringIO
import hashlib
import torch
from torch.nn.functional import softmax
import numpy as np

# ==== Streamlit Config ====
st.set_page_config(page_title="Debate Analyzer", layout="wide")
st.title("Debate Analyzer")

# === Uniform figure size for all charts ===
FIGSIZE_UNI = (5.6, 3.6)

# ==== NLTK setup ====
try:
    nltk.data.find("tokenizers/punkt")
except LookupError:
    nltk.download("punkt")
    nltk.download("punkt_tab")

# ==== Session state ====
for key, default in {
    "show_results": False,
    "analysis_text": "",
    "restored_text": "",
    "text_hash": None,
    "fallacy_threshold": 0.8,
}.items():
    if key not in st.session_state:
        st.session_state[key] = default

# ------------------- TABS -------------------
tab_analyze, tab_learn = st.tabs(["Analyze", "Learn"])

# ===============  TAB 1: ANALYZE (existing) ===============
with tab_analyze:
    # ==== Input section ====
    col1, col2 = st.columns(2)
    with col1:
        st.subheader("Input Transcript")
        typed_text = st.text_area("Transcript:", placeholder="Paste or type here...")
    with col2:
        st.subheader("or Upload TXT File")
        uploaded_file = st.file_uploader("Upload a .txt file", type=["txt"])

    # Threshold selector
    st.subheader("Fallacy Detection Settings")
    st.session_state.fallacy_threshold = st.slider(
        "Fallacy Score Threshold",
        min_value=0.0, max_value=1.0, step=0.05,
        value=0.8, format="%.2f"
    )

    candidate_text = ""
    if uploaded_file:
        try:
            candidate_text = uploaded_file.read().decode("utf-8").strip()
        except UnicodeDecodeError:
            candidate_text = uploaded_file.read().decode("latin-1").strip()
    elif typed_text:
        candidate_text = typed_text.strip()

    if st.button("Analyze", type="primary", disabled=not bool(candidate_text)):
        st.session_state.analysis_text = candidate_text
        st.session_state.show_results = True
        st.rerun()
```

Figure 135: Sample Code - Debate Analyser Development

```python
# ==== Cached Model Loaders ====
@st.cache_resource(show_spinner=False)
def load_punct_model():
    from deepmultilingualpunctuation import PunctuationModel
    return PunctuationModel()

@st.cache_resource(show_spinner=False)
def load_arg_classifier():
    from transformers import AutoTokenizer, AutoModelForSequenceClassification
    model_name = "Sauyang/argument-labelling-deberta-v3"  # Hugging Face model
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForSequenceClassification.from_pretrained(model_name)
    model.eval()
    if torch.cuda.is_available():
        model.cuda()
    return tokenizer, model

@st.cache_resource(show_spinner=False)
def load_fallacy_pipeline(
    model_src: str = "q3fer/distilbert-base-fallacy-classification",
    prefer_local_dir: str = "models/fallacy"
):
    from transformers import pipeline
    from pathlib import Path

    if Path(prefer_local_dir).is_dir():
        return pipeline(
            "text-classification",
            model=str(prefer_local_dir),
            tokenizer=str(prefer_local_dir),
            use_safetensors=True,
            device=0 if torch.cuda.is_available() else -1
        )
    return pipeline(
        "text-classification",
        model=model_src,
        tokenizer=model_src,
        device=0 if torch.cuda.is_available() else -1,
        torch_dtype=None
    )

# ==== Constants ====
VALID_LABELS = ['C', 'G', 'W/Q', 'OTH']
IDX2LABEL = {i: lab for i, lab in enumerate(VALID_LABELS)}

# ==== Processing ====
if st.session_state.show_results and st.session_state.analysis_text:
    raw_text = st.session_state.analysis_text
    current_hash = hashlib.md5(raw_text.encode("utf-8")).hexdigest()

    if st.session_state.text_hash != current_hash or not st.session_state.restored_text:
        with st.spinner("Restoring punctuation..."):
            restored = load_punct_model().restore_punctuation(raw_text)
        st.session_state.restored_text = restored
        st.session_state.text_hash = current_hash

    final_text = st.session_state.restored_text
    threshold = st.session_state.fallacy_threshold

    st.markdown("---")
    st.header("Results")

    sentences = nltk.sent_tokenize(final_text, language="english")
    st.write(f"Your debate transcript contains {len(sentences)} sentences!")
```

Figure 136: Sample Code - Debate Analyser Development

```
# Argument classification
with st.spinner("Classifying argument types..."):
    tokenizer, model = load_arg_classifier()
    preds, confs = [], []
    batch_size = 16

    def to_device(batch):
        return {k: v.cuda() for k, v in batch.items()} if torch.cuda.is_available() else batch

    for i in range(0, len(sentences), batch_size):
        enc = tokenizer(
            sentences[i:i+batch_size],
            padding=True, truncation=True,
            return_tensors="pt", max_length=256
        )
        enc = to_device(enc)
        with torch.no_grad():
            probs = softmax(model(**enc).logits, dim=1)
            preds.extend(torch.argmax(probs, dim=1).cpu().tolist())
            confs.extend(probs.max(dim=1).values.cpu().tolist())

arg_labels = [IDX2LABEL[i] for i in preds]
arg_conf = [round(c, 4) for c in confs]

# Fallacy detection
with st.spinner("Detecting logical fallacies..."):
    fallacy_pipe = load_fallacy_pipeline()
    fallacy_results = fallacy_pipe(
        sentences, truncation=True, padding=True, max_length=256, batch_size=16
    )

def _pick(res):
    if isinstance(res, list):
        res = res[0]
    return res["label"], float(res["score"])

fallacy_labels, fallacy_scores = zip(*[_pick(r) for r in fallacy_results]) if sentences else ([], [])

# Combine results
df = pd.DataFrame({
    "No.": range(1, len(sentences) + 1),
    "sentence": sentences,
    "argument_type": arg_labels,
    "confidence": arg_conf,
    "fallacy": list(fallacy_labels),
    "fallacy_score": [round(s, 4) for s in fallacy_scores]
})
df["fallacy_final"] = np.where(df["fallacy_score"] >= threshold, df["fallacy"], "None")

# ==================== TABLE ====================
st.subheader("Labelled Sentences: Let's see what you have spoken out!")
display_df = df[["No.", "sentence", "argument_type", "fallacy_final"]].rename(
    columns={"fallacy_final": "fallacy"}
)
st.dataframe(display_df, use_container_width=True, hide_index=True)

# KPIs
total = len(df)
any_fallacy = (df["fallacy_final"] != "None").sum()
st.markdown(
    f"**Summary:** {total} sentences • {any_fallacy} with validated fallacies "
    f"({(any_fallacy/total*100 if total else 0):.1f}%)."
)

# ==================== Charts ====================
st.subheader("Charts")

# ROW 1
r1c1, r1c2 = st.columns(2)
with r1c1:
    fig_wc, ax_wc = plt.subplots(figsize=FIGSIZE_UNI)
    wc = WordCloud(width=900, height=360, background_color="white",
                   stopwords=STOPWORDS).generate(final_text)
    ax_wc.imshow(wc, interpolation="bilinear")
    ax_wc.axis("off")
    ax_wc.set_title("Word Cloud", fontsize=12, pad=6)
    st.pyplot(fig_wc, use_container_width=True)
```

Figure 137: Sample Code - Debate Analyser Development

```
with r1c2:
    sent_word_lengths = df["sentence"].apply(lambda s: len(s.split()))
    fig_len, ax_len = plt.subplots(figsize=FIGSIZE_UNI)
    ax_len.hist(sent_word_lengths, bins=20)
    ax_len.set_title("Sentence Length Distribution", fontsize=12, pad=6)
    ax_len.set_xlabel("Length (words)")
    ax_len.set_ylabel("Frequency")
    st.pyplot(fig_len, use_container_width=True)

# ROW 2
r2c1, r2c2 = st.columns(2)
with r2c1:
    arg_counts = df["argument_type"].value_counts().sort_index()
    labels = arg_counts.index.tolist()
    sizes  = arg_counts.values.tolist()
    pie_colors = ["#4C78A8", "#F58518", "#54A24B", "#E45756"]

    def autopct_with_count(pct):
        total = sum(sizes)
        count = int(round(pct/100.0 * total))
        return f"{pct:.0f}%\n({count})"

    fig_pie, ax_pie = plt.subplots(figsize=FIGSIZE_UNI)
    ax_pie.pie(sizes, labels=labels, autopct=autopct_with_count,
            startangle=90, colors=pie_colors)
    ax_pie.axis("equal")
    ax_pie.set_title("Argument Type Distribution", fontsize=12, pad=6)
    st.pyplot(fig_pie, use_container_width=True)

with r2c2:
    ct = pd.crosstab(df["argument_type"], df["fallacy_final"])
    fig_hm, ax_hm = plt.subplots(figsize=FIGSIZE_UNI)
    im = ax_hm.imshow(ct.values, aspect="auto", cmap="Blues", vmin=0)
    ax_hm.set_xticks(range(ct.shape[1]))
    ax_hm.set_xticklabels(ct.columns, rotation=30, ha="right", fontsize=8)
    ax_hm.set_yticks(range(ct.shape[0]))
    ax_hm.set_yticklabels(ct.index, fontsize=8)
    ax_hm.set_xlabel("Fallacy")
    ax_hm.set_ylabel("Argument Type")
    for i in range(ct.shape[0]):
        for j in range(ct.shape[1]):
            ax_hm.text(j, i, str(ct.values[i, j]), ha="center", va="center", fontsize=8)
    fig_hm.colorbar(im, ax=ax_hm, shrink=0.8).set_label("Count")
    ax_hm.set_title("Argument Type x Fallacy Heatmap", fontsize=12, pad=6)
    st.pyplot(fig_hm, use_container_width=True)

# ROW 3
r3c1, r3c2 = st.columns(2)
with r3c1:
    fall_counts = df.loc[df["fallacy_final"] != "None", "fallacy_final"].value_counts()
    fig_fd, ax_fd = plt.subplots(figsize=FIGSIZE_UNI)
    if len(fall_counts) == 0:
        ax_fd.text(0.5, 0.5, "No validated fallacies", ha="center", va="center")
        ax_fd.axis("off")
    else:
        ax_fd.bar(fall_counts.index, fall_counts.values)
        for i, v in enumerate(fall_counts.values):
            ax_fd.text(i, v, str(v), ha="center", va="bottom", fontsize=8)
        ax_fd.set_xlabel("Fallacy")
        ax_fd.set_ylabel("Count")
        ax_fd.tick_params(axis="x", rotation=30, labelsize=8)
    ax_fd.set_title(f"Fallacy Distribution (≥ {threshold})", fontsize=12, pad=6)
    st.pyplot(fig_fd, use_container_width=True)

with r3c2:
    fig_fs, ax_fs = plt.subplots(figsize=FIGSIZE_UNI)
    ax_fs.hist(df["fallacy_score"], bins=20)
    ax_fs.axvline(threshold, linestyle="--")
    ax_fs.set_title("Fallacy Score Distribution", fontsize=12, pad=6)
    ax_fs.set_xlabel("Fallacy Score")
    ax_fs.set_ylabel("Frequency")
    st.pyplot(fig_fs, use_container_width=True)
```

Figure 138: Sample Code - Debate Analyser Development

```
# ------------------ Gemini (kept as in your version) ------------------
  import google.generativeai as genai

  GEMINI_API_KEY = st.secrets["gemini"]["api_key"]
  genai.configure(api_key=GEMINI_API_KEY)

  advice = ""
  def get_gemini_recommendations(df, model_name="gemini-2.0-flash"):
    summary = []
    for _, row in df.iterrows():
      summary.append(f"Sentence: {row['sentence']}\n"
                f"Argument Type: {row['argument_type']}\n"
                f"Fallacy: {row['fallacy_final']}\n")
    summary_text = "\n".join(summary[:50])

    prompt = f"""
    You are an experienced debate coach. Review the transcript with identified argument types and
fallacies.

    Instructions:
    - Focus only on the weaknesses and issues in this transcript.
    - Give direct, professional, and actionable advice to improve (the argument type flow, style, fallacy
content, etc).
    - Be concise and critical: use short sentences or bullet points.
    - Avoid general textbook explanations, only refer to the transcript problems.
    - Maximum 6 bullet points

    Transcript analysis:
    {summary_text}
    """
    model = genai.GenerativeModel(model_name)
    response = model.generate_content(prompt)
    return response.text if response else "No recommendations generated."

  if GEMINI_API_KEY:
    st.subheader("Gemini Flash 2.0 Recommendations")
    with st.spinner("Generating recommendations..."):
      advice = get_gemini_recommendations(df)
    st.write(advice)

  # ------------------ Download ------------------
  st.subheader("Download")
  ordered_cols = ["No.", "sentence", "argument_type", "confidence",
            "fallacy", "fallacy_score", "fallacy_final"]
  csv_buf = StringIO()
  df[ordered_cols].to_csv(csv_buf, index=False)

  c1, c2 = st.columns(2)
  with c1:
    st.download_button("Download Labelled Sentences",
                data=csv_buf.getvalue(),
                file_name="analyzed_transcript.csv",
                mime="text/csv")

  with c2:
    st.download_button("Download Gemini Feedback",
                data=advice,
                file_name="gemini_feedback.txt",
                mime="text/plain")

else:
  st.info("Type or upload text, then click **Analyze**.")
```

Figure 139: Sample Code - Debate Analyser Development

Appendix G: Project Sources Link

Source	Link
Debate Analyser System	https://debate-analyzer.streamlit.app/
Deployment Source	https://github.com/Sauyang520/debate-analyzer
DeBERTa Argument Type Classifier model	https://huggingface.co/Sauyang/argument-labelling-deberta-v3
Implementation Source Code and Processed Dataset	https://github.com/Sauyang520/debate-analyser-space

Table 13: Source Link

Appendix H: Turnitin Similarity Report

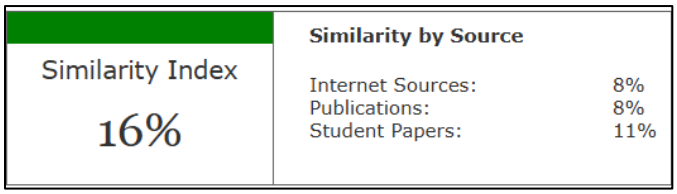


Figure 140: FYP Similarity

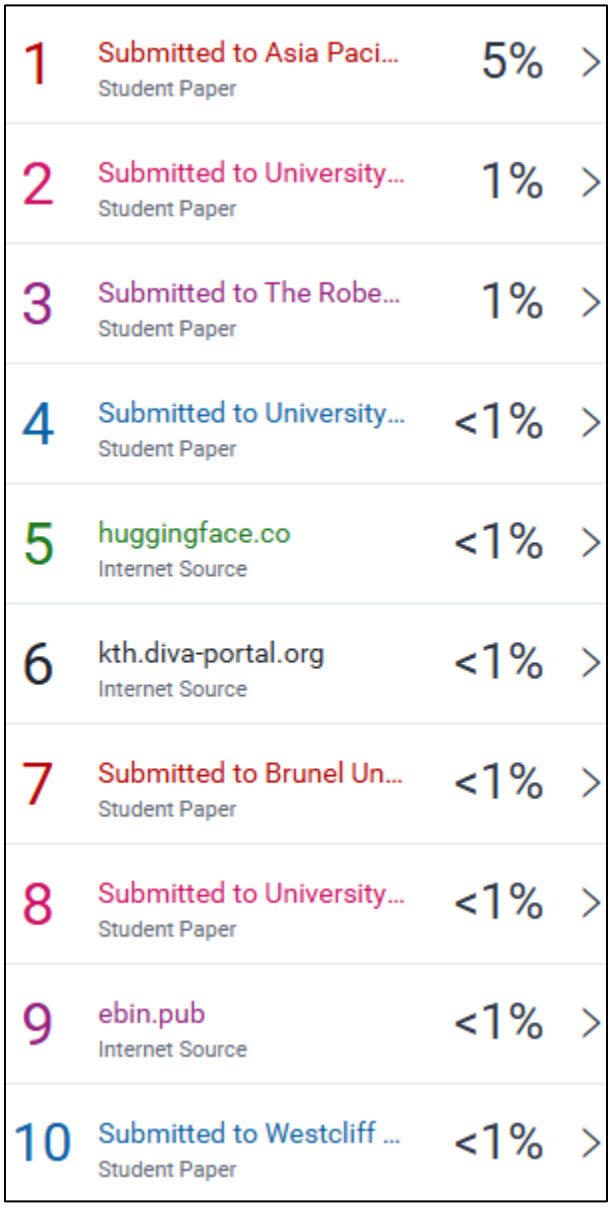


Figure 141: FYP Similarity

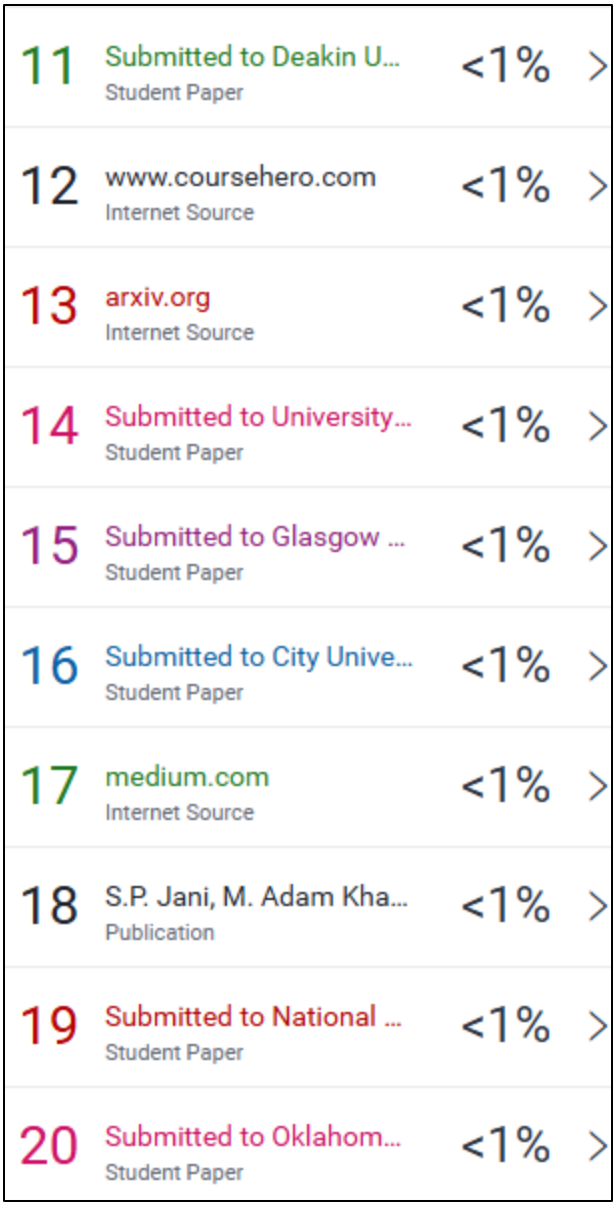


Figure 142: FYP Similarity