Propaganda Detection System

Mohamed Sabry Abdelghany	21100808
Ahmed Mohamed Mahdi	21100822
Abdallah Abdelmalik	21100833

Table of Contents:

1.Abstract:

This research explores the application of advanced natural language processing (NLP) models for the detection of propaganda in text, focusing on two distinct approaches: AraBERT for Arabic language processing and a BERT-GPT4_Augmented model. We evaluate these models on binary and multi-label classification tasks using separate datasets. The binary classification dataset categorises text as either propagandistic or non-propagandistic, while the multi-label dataset identifies various propaganda techniques.

Both models were fine-tuned using Python-based frameworks, incorporating techniques such as tokenization, stop-word removal, and normalisation for preprocessing the text data. The AraBERT model exhibited strong performance in binary classification, achieving high precision and F1 scores, while the BERT-GPT4_Augmented model demonstrated superior recall, capturing a broader range of propagandistic content.

For the multi-label classification task, the AraBERT model's performance was lower, revealing the additional complexities involved in detecting multiple propaganda techniques. This research provides valuable insights into the effectiveness of modern NLP models in automated content moderation and information integrity, highlighting the need for further exploration in multi-label classification challenges and model optimization.

2.Introduction

2.1Overview

The spread of disinformation and propaganda in digital media has escalated, creating an urgent need for automated detection systems. This project aims to address this issue by developing an AI-powered propaganda detection system that leverages advanced natural language processing (NLP) models to classify and detect propaganda in text. We evaluate two state-of-the-art models: AraBERT, tailored for Arabic language processing, and BERT-GPT4_Augmented, which integrates BERT with GPT-4 for enhanced language understanding.

In addition to the NLP models, the project involves the development of a frontend using React.js and a backend built with Flask (Python). The frontend enables users to input text and receive real-time feedback on whether the text contains propaganda. The backend processes user inputs, passes them to the appropriate model, and returns the classification results to the frontend.

2.2Problem Statement

Propaganda remains a significant tool for influencing public opinion, manipulating facts, and disseminating misleading information. Detecting such content efficiently is crucial, especially in the era of mass media and social platforms where propaganda can spread rapidly. Manually identifying propaganda is not feasible at scale, leading to the need for automated solutions using AI models. The project addresses this problem by utilizing modern NLP techniques for detecting propaganda, offering both binary classification (propaganda vs. non-propaganda) and multi-label classification (specific propaganda techniques). Additionally, we implement a full-stack web application to make this detection process accessible and user-friendly.

2.3Motivation

The unchecked spread of propaganda and disinformation poses a threat to public discourse, undermining democratic processes and fostering social division. Automated detection systems offer a scalable solution to identify and mitigate the impact of such content. By building a web-based AI solution, this project contributes not only to the field of propaganda detection but also to real-world applications in content moderation, media monitoring, and information integrity.

The use of AraBERT for Arabic language content and the BERT-GPT4_Augmented model provides unique opportunities to evaluate how well these models handle complex language tasks, making this project relevant for regions with high volumes of propaganda in Arabic media. The integration of the models into a web-based platform using React.js and Flask allows for real-time interaction, making AI-driven propaganda detection more accessible to users.

2.4Scope

This project involves two key areas: AI model development and web application deployment. The scope includes:

- NLP Models
- : Fine-tuning and evaluating AraBERT and BERT-GPT4_Augmented models for propaganda detection. The models are applied to two tasks:
    - Binary Classification: Classifying text as propagandistic or non-propagandistic.
    - Multi-Label Classification: Identifying various propaganda techniques within a single text sample.
- Frontend (React.js): A user-friendly interface where users can input text and receive classification results (Yes/No) or a breakdown of propaganda techniques. The frontend also displays the overall accuracy and performance of the model.
- Backend (Flask, Python): A server that handles user requests, processes the input text, and communicates with the NLP models to generate predictions. The backend ensures smooth interaction between the user and the AI models.
- Deployment: The project will be deployed as a web-based tool, allowing users to access the propaganda detection system from any device with internet connectivity.

3.System Architecture

The system architecture for this project consists of three primary components: the Frontend (React.js UI), Backend (Flask Server), and the Model Integration for AI-based propaganda detection. These components work together to form a complete web-based solution that allows users to interact with the NLP models in real-time, classify input text, and detect propaganda with high accuracy. Below is a breakdown of each system component.

3.1 Frontend (React.js UI)

The frontend of the system is built using React.js, a modern JavaScript library for building user interfaces. The frontend provides a user-friendly interface where users can input text that they want to analyze for propaganda. The key features of the React.js frontend include:

- Text Input Field: A text box where users can enter the content they want to analyze.
- Submit Button: Once the text is entered, the user submits it by clicking the button, triggering the interaction with the backend.
- Result Display: After processing, the classification result is displayed in real-time. If the text contains propaganda, it shows a "Yes" or "No" for binary classification, or specific propaganda techniques for multi-label classification.
- Responsiveness: The UI is designed to be responsive, ensuring seamless use on both desktop and mobile devices.
- Real-Time Feedback: Results are updated in real time, allowing users to quickly see the outcome of their analysis.

The frontend communicates with the backend server using API calls, ensuring that the user interaction is smooth and efficient.

3.2 Backend (Flask Server)

The backend of the system is implemented using Flask, a lightweight Python web framework. The backend serves as the bridge between the frontend UI and the AI models, handling user requests and managing the interactions with the models. The primary responsibilities of the Flask server include:

- API Endpoints: The backend exposes RESTful API endpoints that the frontend uses to send the input text and receive the prediction results. A common endpoint structure might include routes like /predict for binary classification.
- Text Preprocessing: Before passing the text to the NLP models, the backend handles text preprocessing (e.g., tokenization, normalization) to ensure that the data is formatted correctly for model input.
- Model Loading and Management: The backend manages loading and initializing the AraBERT and BERT-GPT4_Augmented models. Flask ensures that the models remain accessible in memory for efficient, repeated predictions without needing to reload the model for each request.
- Response Handling: After the AI models generate their predictions, the backend formats the output and sends the result back to the frontend in JSON format. This allows the frontend to dynamically update the UI with the results.
- Scalability: Flask's lightweight nature ensures that the backend can be scaled easily, handling multiple user requests simultaneously without performance bottlenecks.

3.3 Model Integration

At the core of the system are the NLP models that perform propaganda detection. The models are fine-tuned for both binary and multi-label classification tasks. The integration of these models with the backend is crucial for real-time performance and accuracy. Here's how the model integration works:

- AraBERT Model: This model is fine-tuned specifically for Arabic language propaganda detection. It is integrated into the backend to handle both binary and multi-label classification tasks. AraBERT takes in the preprocessed Arabic text and outputs either a binary label (Yes/No) for the presence of propaganda or multiple labels for various propaganda techniques.

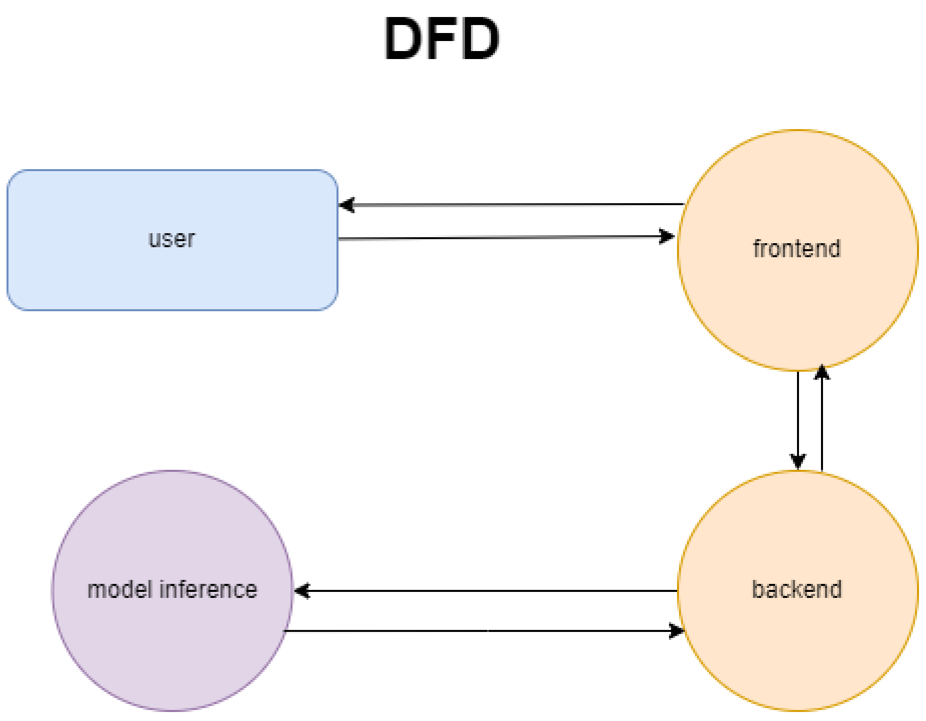
- BERT-GPT4_Augmented Model: This model combines the strengths of BERT and GPT-4 to provide enhanced performance in understanding complex language structures. It is particularly useful for multi-label classification where the text might exhibit multiple propaganda techniques simultaneously. The backend integrates this model to handle more nuanced and complex text classifications.
- Model Selection Logic: Depending on the user input (language and classification type), the backend decides which model to use for prediction. For binary classification in Arabic, AraBERT is selected, whereas for multi-label classification tasks or enhanced results, the BERT-GPT4_Augmented model is utilized.
- Model Optimization: To ensure efficient performance in real-time, both models are optimized for inference, with preloaded weights and minimized latency. Flask's backend ensures that the models are invoked efficiently, reducing response times for the user.

3.4 Workflow Overview

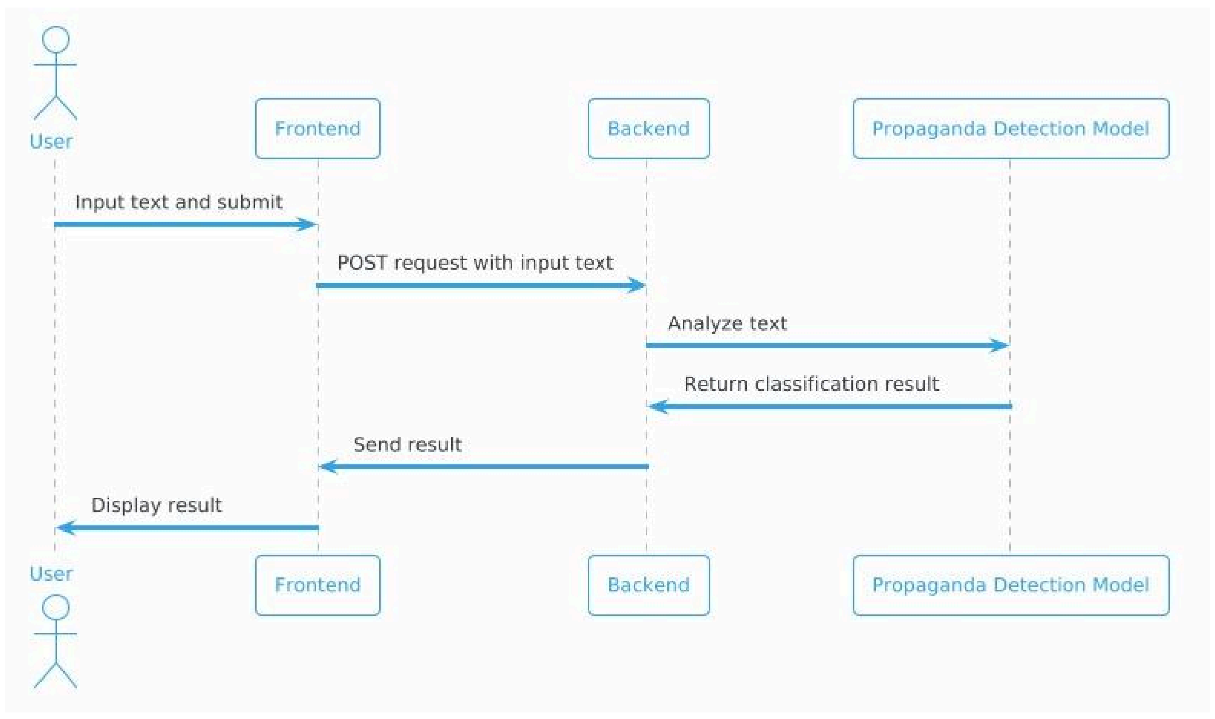
- User Input: The user submits text through the frontend UI.
- Frontend-Backend Communication: The frontend sends the input text to the backend via API calls.
- Text Processing and Model Prediction: The backend preprocesses the text, selects the appropriate NLP model (AraBERT or BERT-GPT4_Augmented), and generates predictions.
- Result Delivery: The backend formats the prediction results and sends them back to the frontend.
- Display: The frontend displays the result to the user, showing whether the text contains propaganda or which specific propaganda techniques were identified.

4. Diagrams

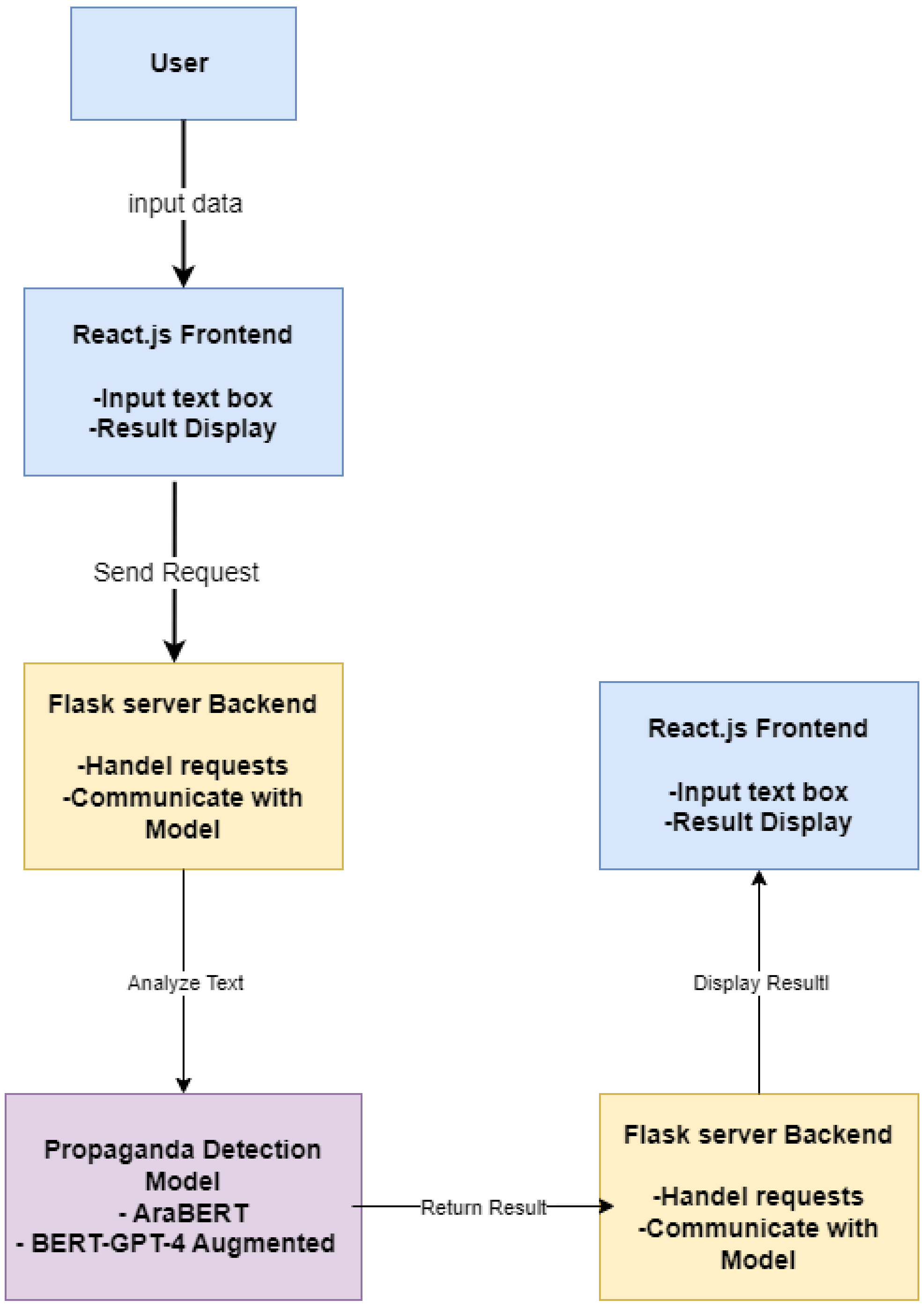
4.1.Data Flow Diagram



4.2.Sequnece Diagram


User
Frontend
Backend
Propaganda Detection Model
Input text and submit
POST request with input text
Analyze text
Return classification result
Send result
Display result
User
Frontend
Backend
Propaganda Detection Model

4.3.Archeticture System Diagram



5. Propaganda Detection Model

5.1 Model Selection and Justification

- The study compares two models: AraBERT and a BERT-GPT-4 Augmented model.

- AraBERT: Chosen for its strong performance in Arabic language processing, it's tailored for tasks like propaganda detection, especially in binary classification tasks.
- BERT-GPT-4 Augmented Model: Combines the architecture of BERT with the capabilities of GPT-4 to enhance performance, particularly in capturing various propaganda techniques. This model demonstrates higher recall, suggesting it is better at identifying all relevant propaganda instances but with slightly lower precision.

5.2 Model Training and Dataset

- Binary Classification Dataset: Consists of 8000 articles, divided into 6002 for training, 672 for validation, and 1326 for testing. The model was trained using binary cross-entropy loss and optimized with the Adam optimizer.
- Multi-Label Classification Dataset: This dataset contains articles labeled with multiple propaganda techniques. The model for this task struggled to achieve the same level of performance as in binary classification, indicating the complexity of the task.
- Training: Both models were fine-tuned for their respective tasks with metrics like accuracy, precision, recall, and F1-score used for evaluation**.**

5.3 Model Deployment and API Integration

- The models were initialized and fine-tuned for both binary and multi-label classification tasks.
- The **AraBERT** model showed robust performance in binary classification, while the **BERT-GPT-4 augmented model** demonstrated better recall but with some limitations in precision.
- Challenges were noted with accessing the GPT-4 model due to system errors and delays, which impacted the overall deployment and API integration process

6. System Workflow

6.1 User Interaction with the System

- The user interacts with the system through a React.js-based UI.
- The interface includes a text box where users input text they want to analyze for propaganda detection.
- After entering the text, the user submits the input (likely by clicking a button).
- This triggers a request to the backend (Flask server), which processes the input for analysis.

6.2 Data Flow Between Client and Server

1. **Frontend (Client Side):**
   - User input (text) is captured from the text box in the React.js interface.
   - The input text is sent via a POST request from the frontend to the backend (Flask server).
2. **Backend (Server Side):**
   - The Flask server receives the POST request containing the input text.
   - The server processes the text by passing it to the Propaganda Detection Model (e.g., AraBERT or BERT-GPT-4 Augmented).
   - The detection model analyzes the text and classifies it as either propagandistic or non-propagandistic (or assigns multiple labels if it's a multi-label classification task).
   - The server returns the detection result (likely as a JSON response) to the frontend.

6.3 Response Handling and Result Display

Once the Flask server sends the analysis results back to the frontend:

- The React.js client processes the JSON response and extracts the necessary information (e.g., labels, confidence scores).

- The results (whether binary classification or multi-label detection) are displayed in a designated area of the user interface.
- This result could include a message indicating whether the text is propagandistic, alongside additional data such as specific propaganda techniques detected (in case of multi-label classification) or confidence scores.

7. Testing and Validation

7.1 Frontend Testing

- Usability Testing: Ensured user-friendly interaction with the React.js UI across devices.
- Cross-Browser Testing: Verified consistent performance on Chrome, Firefox, Safari, and Edge.
- Responsiveness: Tested layout on desktop, tablet, and mobile screens using Bootstrap.
- Form Validation: Checked for valid inputs and edge cases like empty or overly long text.
- Real-Time Feedback: Ensured minimal delay in displaying results after form submission.

7.2 Backend Testing (API Responses)

- Unit Testing: Tested Flask functions for text preprocessing and model interaction.
- API Testing:
  - Validated input handling, response format, and status codes using Postman and Pytest.
  - Ensured proper error handling for invalid requests.
- Load Testing: Simulated multiple API calls to check backend performance under stress.
- Latency Testing: Measured API response times to ensure real-time result delivery.

7.3 Model Accuracy and Performance Testing

- Binary Classification: Evaluated using accuracy, precision, recall, and F1-score.
- Multi-Label Classification: Assessed model performance across multiple labels using BCEWithLogitsLoss.
- Cross-Validation: Used k-fold cross-validation to avoid overfitting.
- Inference Time: Optimized model response time for quick predictions.
- Comparative Analysis: Compared AraBERT and BERT-GPT4_Augmented for both binary and multi-label tasks.

8. Challenges and Solutions

8.1 Frontend Development Challenges

- User Interface Consistency: Ensuring a uniform design across different devices and browsers was challenging.
  - Solution: Utilized Bootstrap for responsive design and conducted thorough cross-browser testing to ensure consistency.
- Real-Time Feedback: Achieving low latency in displaying results from the backend was initially problematic.
  - Solution: Optimized API calls and reduced data processing on the client side to ensure quick updates.
- Input Validation: Handling user inputs effectively to prevent errors and improve user experience was a concern.
  - Solution: Implemented robust form validation and error messages to guide users in real-time.

8.2 Backend and Model Integration Issues

- Model Loading Latency: Initial load times for the NLP models were slow, affecting user experience.
  - Solution: Kept models pre-loaded in memory on the Flask server to minimize loading times during requests.
- API Response Handling: Managing errors from the models or unexpected input formats led to crashes.

    - Solution: Implemented comprehensive error handling and logging to catch and address issues gracefully without impacting user experience.
- Data Preprocessing: Ensuring consistent preprocessing between frontend submissions and backend processing was challenging.
    - Solution: Standardized preprocessing functions to ensure uniformity across all text inputs before model inference.

8.3 Performance Optimization

- Model Inference Speed: Slow inference times impacted the overall responsiveness of the application.
    - Solution: Optimized model architecture and used techniques like mixed precision training to reduce computational load.
- API Throughput: The backend struggled under heavy loads with simultaneous requests.
    - Solution: Implemented caching strategies for frequently requested data and load testing to identify bottlenecks, resulting in improved throughput.
- Frontend Load Times: Initial page load times were longer than desired, affecting user experience.
    - Solution: Reduced the size of frontend assets and implemented lazy loading for non-critical components to enhance performance.

9. Conclusion

9.1 Summary of Achievements

- Successfully developed a Propaganda Detection System using advanced NLP models (AraBERT and BERT-GPT-4 Augmented).
- Implemented a React.js frontend that allows users to input text and receive real-time analysis on whether the content contains propaganda.
- Built a Flask backend that efficiently processes input text, communicates with the detection model, and returns results to the client.

- Demonstrated effective propaganda detection, with the AraBERT model excelling in binary classification and the BERT-GPT-4 Augmented model achieving high recall in multi-label classification tasks.
- Overcame various technical challenges, including integrating the models, handling API requests, and ensuring the system performs well with real-world data.

9.2 Lessons Learned

- Model Selection Matters: While both AraBERT and BERT-GPT-4 Augmented models provided good results, model performance varied depending on the complexity of the task (binary vs. multi-label classification). This emphasizes the importance of choosing the right model for specific tasks.
- API Integration Complexity: Deploying the models and ensuring smooth client-server communication can be complex, especially when dealing with larger models (e.g., GPT-4 integration). These challenges include system errors, access issues, and latency in responses.
- User Experience Considerations: Building an intuitive and responsive frontend is crucial for user engagement. Ensuring that the results are displayed clearly and quickly improves the overall system usability.

9.3 Final Thoughts

- The *Propaganda Detection System* has the potential to be a powerful tool for media monitoring, social media analysis, and promoting information integrity. By leveraging state-of-the-art language models, the system can detect and expose manipulative content, thereby helping to combat misinformation.
- Future improvements could involve expanding the model's capabilities to detect other forms of manipulative content (e.g., fake news, biased reporting) and improving multi-label classification performance.
- Overall, this project demonstrates the effectiveness of integrating advanced NLP models with a modern web-based architecture, providing a foundation for further enhancements and applications in the field of automated content moderation.

THANKS!