

# DIT826 – Software Engineering for Data-Intensive AI Applications

## Group 3 CyberSafeAI



### [CyberSafeAI - Content Moderation System](#)

Name	Email
Jitish Rajankumar Padhya	<a href="mailto:guspadji@student.gu.se">guspadji@student.gu.se</a>
M.Ali Elhasan	<a href="mailto:guselhmu@student.gu.se">guselhmu@student.gu.se</a>
Nishchya Arya	<a href="mailto:gusaryni@student.gu.se">gusaryni@student.gu.se</a>
Raghav Tengse	<a href="mailto:gustengra@student.gu.se">gustengra@student.gu.se</a>
Utkarsh Singh	<a href="mailto:gussinut@student.gu.se">gussinut@student.gu.se</a>

## Individual Contribution Form (more detail in appendix II)

Individual Name	Description of Contribution	Signature
Jitish Rajankumar Padhya	<p><b>User-side frontend (moderation app)</b> (creating layout, components, fixing logics, code documentation, README)</p> <p><b>CI/CD</b> (creating gitlab pipeline to allow for building, testing and deployment. Also, working on understanding best solution for deployment on GCP using buckets and kubernetes in pair programming environment)</p> <p><b>Refactoring, restructuring, and bug fixes of moderation app and CSA_AdminApp</b> (Architectural decision of splitting and rerouting the apps into directories of frontend, backend, ai_model)</p> <p><b>Removing toyModel and its dependencies</b> (to make the application more lightweight)</p> <p><b>Experimentation and feature fixes of explainable AI</b> (Use of LIME (removed), BERTtokenizers, highlighting all words that reflect the probability of all 6 categories of toxicity)</p> <p><b>Report writing</b> (Section 1 (problem statement and system descriptions, goals and functionality), Section 2 (data), Section 3 (features and inference models), Section 7 (advanced AI functionality),</p> <p><b>Architecture diagrams</b> (deployment, component, and ML pipeline diagrams)</p> <p><b>Analysis history (user-side)</b> (worked on filtering, fetching, and bugs regarding number rounding of the probabilities (analysis history was removed in the final iteration of the app))</p> <p><b>Main and AI_model directory READMEs</b></p>	Jitish
M.Ali Elhasan	<p><b>Toy-Model:</b> Creating a full toy model included the AI model and GUI.</p> <p><b>Project Structure:</b> Prepare the project structure and separate the folder direction to have an efficient and professional development process.</p>	M.Ali

	<p><b>GUI:</b> Set up the user frontend and prepare the html file to be modified easily by team members.</p> <p><b>AI model:</b> Have expertised implementing different AI models, Supervised and Unsupervised types, for the toy model we had a light version, and concern to our system idea we had to implement a NLP based model.</p> <p><b>Train model:</b> Apply the GPU training process to have a faster training process and to have better results when users use our system.</p> <p><b>CI/CD:</b> Modified the deployment process to have an efficient training process locally and online.</p> <p><b>DevOps:</b> Get more experience in setup server and handle debugs and track the logs to diagnose the server side.</p> <p><b>Report checking:</b> Had to check the report rarely after each step with the team.</p> <p><b>Backend:</b> Code documentation</p>	
Nishchya Arya	<p><b>Admin-side frontend (CSA_AdminApp)</b> (creating and styling layout, database handling for admin app, download and upload functionality)</p> <p><b>Docker</b> (Created and managed the Dockerfile, streamlining the containerized setup for Django development and deployment.)</p> <p><b>Deployment</b> (Created and managed the Dockerfile, deployment.yaml, service.yaml, and sqlite-pv-pvc.yaml files to streamline containerization, orchestrate Kubernetes resources, and ensure persistent database storage for the application, while also setting up and managing the entire Google Cloud Platform environment for the project.)</p> <p><b>CI/CD</b> (minor fixes to CI/CD to ensure deployment with every successful pipeline run)</p> <p><b>Refactoring, restructuring, and bug fixes of moderation app and CSA_AdminApp</b> (Architectural decision of splitting and rerouting the apps into directories of frontend, backend, ai_model)</p> <p><b>Backend</b> (code documentation , README)</p>	Nishchya

	<b>Report writing</b> (Section 1 (Defined and outlined the project requirements (both functional and non-functional), Section 4 (Production system), Section 5 (Software quality))  <b>Architecture diagrams</b> (deployment, component, and ML pipeline diagrams)	
Raghav Tengse	<b>Admin-side Frontend</b> ( developing log-in UI and functionality, authentication services, log out functionality)  <b>Creating Admin User Model</b> <b>Testing</b> (generate unit tests for all endpoints in the system, including model creation tests)  <b>Report writing</b> (create user stories for the requirements)	Raghav
Utkarsh Singh	<b>Admin-side Frontend</b> (creating and styling layout, added filters in the database on the basis of labels present in database)  <b>User-side Frontend</b> (displaying all the six labels in the dashboard, styling and layout for text analysis)  <b>Explainable AI</b> (Use of LIME, BERTtokenizers, highlighting all words that reflect the probability of all 6 categories of toxicity)  <b>Report Writing</b> (Defined and outlined the project requirements (both functional and non-functional), Section 6 - Reflection(final report))	Utkarsh

## Section 1. The concept of the AI system

### Problem Statement and System Description

*Author: Jitish*

The level of cyber crime is at an incline due to the possession of social media to a major part of the population. This creates an opportunity to be able to design a system that helps us to identify and flag text based on aspects of toxicity, threat, insults, and more.

CyberSafeAI system allows us to upload text in a given field (can be messages, tweets, etc.) for toxicity analysis. The web-app detects and categorizes the data in the following labels of toxicity: toxic, severe\_toxic, obscene, threat, insult, identity\_hate and the respective percentage to match. To achieve explainable AI, the aim is to have highlighted key problems that are

showing the various traits of toxicity. Additionally, this allows users of various fields some primary targets can include, individuals on social media, content creators, and social media managers to analyze their text before posting it on the targeted platforms. To conclude, the following will lead to a safer environment for users to interact across social media in a healthy way and avoid unintended backlash or issues of such matters.

## Goals and Functionality

*Author: Jitish*

- ❖ Have an automated prescreen of the text prior to posting it on any platform
- ❖ Provide feedback for the outcome of the analysis incorporating explainable AI
- ❖ Avoid any form of adverse reaction due to accidental use of the various labels involved in toxicity
- ❖ Have the ability to update the model to improve the precision and accuracy of the analysis

## Requirements

User side:

RNo.	Requirement	User story
1	The system shall have an interface for the user to upload their content	As a blogger I want to have an interactive interface to submit my content for ethics analysis.
2	The system shall allow user to have real time content analysis	As a blogger i don't want to wait too long to receive results in order to make necessary changes based on timely results
3	The system shall highlight the concerning portions of the text to prompt the user. (Explainability AI)	As a blogger I want to know what parts of my content are offensive or unethical in order to edit the text to be ethically sound.
4	The system shall be able to classify the type of feed in content	As a blogger I want to know the reason why my content is considered offensive in order to ensure that the content does not repeat the issue.
5	The system shall be able to tell the severity of the entered content	As a blogger I want to know the intensity of unethically my content exhibits in order to modify it and ensure ethicality.

Admin side:

R no.	Requirement	User story
1	The system shall have functionality for the admin to log in.	As an administrator I want to securely log in to the system so that I can access administrative functions
2	The system shall have an interface for the admin to check for system usage logs.	As an administrator I want to be able to monitor system usage via system usage logs so that I can identify patterns and ensure proper usage.
3	The system shall allow the admin to upload and integrate new training datasets.	As an administrator I want to be able to post new datasets to retrain the content moderation system
4	The system shall allow the admin to view previous and current model versions	As an administrator, I want to view previous and current model versions so that I can track changes and maintain version control.
5	The system shall allow the admin to rollback to previous model versions if needed.	As an administrator I want to be able to access previous models in order to move back to a stable version if issues arise with the current version.
6	The system shall allow the admin to back up the existing model states.	As an administrator I want to be able to back up to prior model states in order to maintain network stability and restore to prior configurations if needed.

## Non Functional Requirements

### 1. Performance

- **Real-Time Processing:** The system shall provide real-time content analysis for user uploads, with processing times ideally not exceeding 3 seconds per analysis.
- **Response Time:** System responses to requests (such as content classification and severity assessment) shall be under 2 seconds to maintain a smooth user experience.

### 2. Reliability and Availability

- **Uptime:** The system shall maintain 99.9% uptime, providing continuous availability for users and administrators.

### 3. Security

- **Role-Based Access Control:** The system shall enforce strict role-based access controls (RBAC) to segregate admin and user functionalities.

### 4. Usability

- **Consistent User Interface:** The system shall provide a consistent and intuitive interface for both users and administrators, adhering to best practices for UI design.
- **Documentation and Help:** Comprehensive documentation and help resources shall be provided for users and admins, covering all system functionalities

### 5. Maintainability

- **Modular Codebase:** The system's architecture shall support modularity, allowing components to be updated or replaced with minimal impact on the overall system.
- **Code Documentation:** Clear, consistent code documentation shall be maintained for all system components to facilitate maintenance

### 6. Portability and Compatibility

- **Deployment Flexibility:**  
The system shall be containerized (e.g., via Docker) and support deployment on cloud platforms (e.g., Kubernetes) to ensure flexibility and ease of scaling.

### 7. Accuracy and Precision

- **Model Accuracy:**  
The system shall aim for a high degree of classification of the inputted content.
- **Retraining Cycle:** The system shall support retraining of models on new data, ensuring performance improvements are demonstrated before deployment.

### 8. Privacy

- The system shall comply with data protection regulations by anonymizing all user data before storing it in the database.

## Section 2. Data

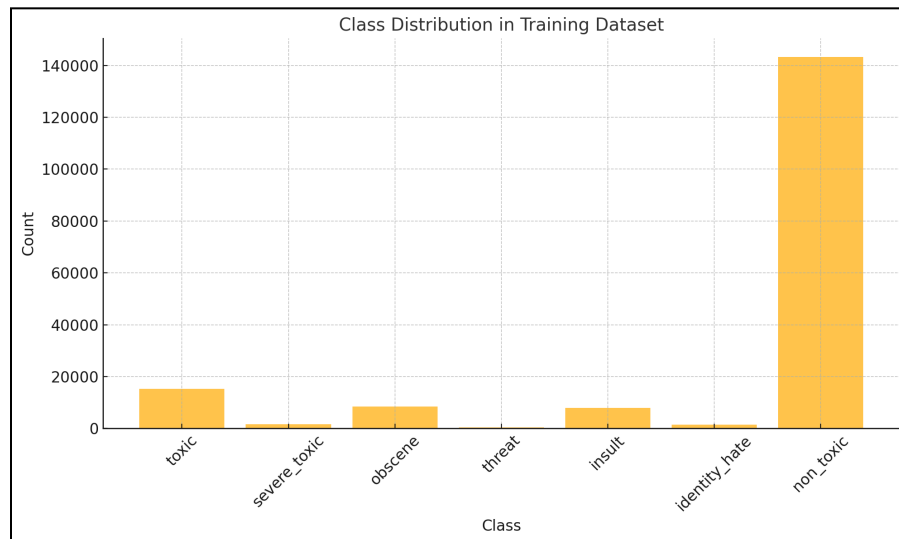
*Author: Jitish*

The dataset is Toxic Comment Classification Challenge and is on Kaggle <sup>1</sup>. It comprises two sets of data one for training the other for testing. The training dataset consists of 159,571 data points the labels columns include: id, comment\_text, toxic, severe\_toxic, obscene, threat, insult, identity\_hate. Where the id and comment\_text would be our features, while the rest are binary target labels (0 meaning not present, 1 meaning present) see *figure 1* for example. One comment can belong to multiple target labels representing more than one type of toxicity. On the other hand the test data includes 153,164 data points only consisting of id and comment\_text column.

id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0000997932d777bf	Explanation	0	0	0	0	0	0
000103f0d9cfb60f	D'aww! He matches this background colour I'm seemingly stuck w.o	0	0	0	0	0	0
000113f07ec002fd	Hey man, I'm really not trying to edit war. It's just that this guy is cc 0	0	0	0	0	0	0
0001b41b1c6bb37e	"	0	0	0	0	0	0
0001d958c54c6e35	You, sir, are my hero. Any chance you remember what page that's 0	0	0	0	0	0	0
00025465d4725e87	"	0	0	0	0	0	0

*Figure 2.1 : Example of training dataset*

The primary goal is to be able to classify inputted text into the given targeted labels to be able to justify the toxicity analysis, though the training data presented provides a significant imbalance as shown in *figure 2*.



*Figure 2.2 : Distribution of training dataset*

Furthermore, this issue is to be tackled through the possibility of 3 ways: oversampling of underrepresented classes, undersampling of overrepresented classes, or higher weight on loss function to the underrepresented classes.

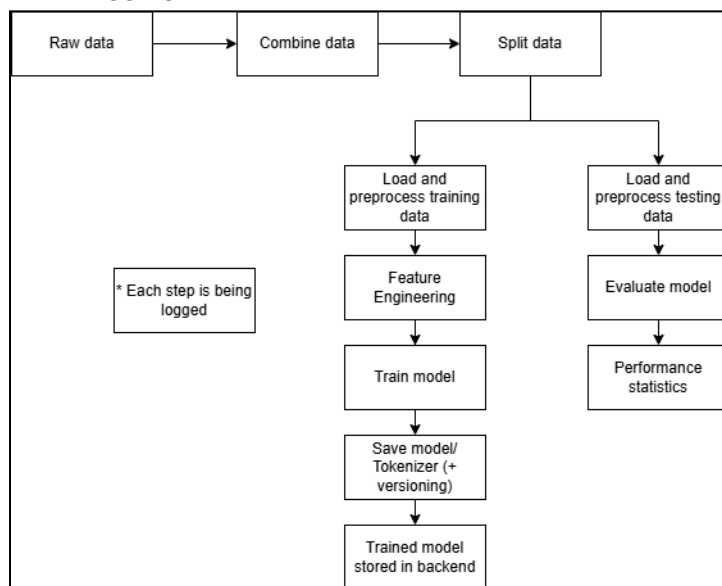
<sup>1</sup> <https://www.kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge/data>



## Section 3. Features and inference model

*Author: Jitish*

The features and inference model is designed with optimisation, accuracy, and precision in mind. The ML pipeline ensures a smooth flow from receiving raw data, training the model, and evaluating it against the test data as seen in *figure 3.1* and described in chronological order below while the *logger.py* logging each step to keep track of the pipeline and easier debugging.



*Figure 3.1 : ML pipeline*

### 3.1 Raw Data, Data Combination, and Splitting Data

The previous dataset is introduced with new users or other sources of data (if exists) for retraining of model. In *combine\_data.py*, the utility handles new data in the form of .csv with the existing .csv file from the previous iteration. Additionally it appends the existing .csv file with the new data though not allowing duplicates to ensure integrity of the data. The updated combined data is then split into training and testing .csv files in *data\_splitter.py*. The split used is 70% training and 30% testing. Furthermore, *random\_state* is used to ensure consistent split of the outputs through random seed for reproducibility.

### 3.2 Data Loader, and Data Preprocessing

The *data\_loader.py* ensures the file is correctly loaded into memory distinguishing train or test data paths using *is\_train* variable. The data is loaded through the use of pandas library storing it as a DataFrame. As well as, the data is checked for if it is empty or not, on success it returns the loaded DataFrame, and on failure returns *None*.

Moving further the preprocessing of the data occurs in *data\_preprocessing.py* for both the training and testing data. The main objective is to remove noise, inconsistencies, or irrelevant text that can affect the pretrained BERT model. These include, put the text in lowercase, remove URLs, remove mentions, and remove non-alphabetical characters.

### 3.3 Target Label Creation

The dataset involves multiple label output (toxic, severe\_toxic, obscene, threat, insult, identity\_hate). Each of these columns indicates whether a comment belongs to a specific toxic category (1) or not (0). The aim to use Label binary classification column is to ensure that the model flags any type of toxicity as toxic. Hence, if any one of the labels are 1 then the Label column will be marked as 1 indicating being toxic; otherwise it remains 0 indicating non-toxic.

### 3.3 Train and Save Model

Finally the training of the model involves the use of *training.py* and *config.py*. The process involves fine-tuning a **Bert-base-uncased** model to classify the multiple labels. The choice of Bert model was due to it being not too heavy-weight while not compromising too much on the performance. We previously tried distilBert which wasn't too accurate and precise though being lightweight. To further increase performance we capped maximum sequence length to 128 tokens to be efficient with processing as being a normal-sized model requiring computational time.

In terms of specific configurations applied are in line with the chosen BERT model to have a balance between performance and efficiency. This includes the following training parameters:

- Batch size of 16 for training and 64 for evaluation per device, to allow memory efficiency especially VRAM, boosted more due to access of CUDA. As, after the given sample size recalculations such as gradient, loss are done.
- Learning rate of  $2e-5$ , the learning rate is so small due to the BERT model already being trained and is well optimised hence only fine-tuning is required so the precision increases and avoids overshooting the optimum point, leading to a stable convergence. Furthermore, 500 warm up steps are in place to stabilize the learning rate.
- Epoch of 3, refers to the model training 3 times on the entire dataset, the number is on the lower end due to being pre-trained hence only requiring fine-tuning to the new task. On the other hand, higher numbers can lead to overfitting of data.

Moving onto the data preparation, it involves 4 key steps: tokenization, label preparation, data conversion, and dynamic padding. BertTokenizer is used to convert raw text into tokens compatible with the model, capping it at 128 for reduced processing time. Then the labels for multi-label classification are extracted from the six toxicity columns and stored as lists of binary values. In the process, the pandas DataFrame is converted to a Hugging Face Dataset due to its compatibility and optimisation for tokenized text data. Finally, dynamic padding (adding "empty" tokens or truncating it) is in place to ensure that all tokenized inputs are of the same length in a batch BERT process. Upon the completion of data preparation the Hugging Face trainer class/API handles the training by gathering the data and parameters including forward, backward pass (backpropagation), logging, and having intermediate checkpoints to save results periodically. To conclude the training, the model along with the tokenizer is then stored versionized inside a separate github repo where the models can be pushed and fetched.

### 3.4 Testing and Evaluation Stage

The testing and evaluation stage involves preparation of the testing data in the same way as the training data. Upon tokenization of the input data, `model.eval()` function ensures it is evaluating and not training, and keeps the output same and consistent. The output consists of logits which are scores for each of the six classes. Additionally, `torch.argmax()` function is used to identify the index of the highest scoring class, indicating the most confident prediction. The predicted label is then stored along with the ground truth for evaluation metrics using scikit-learn; the metrics involve accuracy, precision, recall, and F1 score.

### 3.5 Discussion of ML Pipeline

In terms of the decisions made within the pipeline it consisted of trial and error to discover the best outcome for the versioning of the model to provide high values though balanced as well between the metrics of accuracy, precision, recall, and f1 score. As the image (*figure 3.2*) below shows two of the current versions deployed during the time of completion.

<b>V1:</b> {'accuracy': 0.28947819184491979, 'precision': 0.3811542441431326, 'recall': 0.38947819184491979, 'f1': 0.28511318477545647 }	<b>V2:</b> {'accuracy': 0.8947819184491979, 'precision': 0.811542441431326, 'recall': 0.8947819184491979, 'f1': 0.8511318477545647 }
---	---

*Figure 3.2 : Evaluation Metric for Version 1 and 2*

Version 1 was trained on a smaller dataset consisting of 5,000 total data points, evenly balanced across all labels. While this balanced dataset ensured equal representation of all the labels, the model struggled to generalize common functionality due to the limited size and diversity of examples, leading to a low accuracy of 29% and an F1 score of 28% showing imbalance between precision and recall. These results indicated that the model was not learning the broader patterns necessary to handle real-world text. As often it struggled with identifying words such as “sweet” being flagged toxic due to the limited dataset being provided.

On the other hand, due to the larger sum of non-toxic labels in the complete dataset it is better at generalizing common words as non-toxic. Version 2 utilized the full dataset of 159,000 data points. This resulted in a significant improvement across all metrics, with accuracy reaching 89% and an F1 score of 85%. The larger dataset allowed the model to generalize better especially to common language as noticed specifically in the case of “you”, “sweet”.

To conclude, experimentation of different weights of each label though its effect on non-toxic language being flagged was quite high. Hence, it made sense to use CUDA to accelerate the 159,000 data points due to being technically demanding, completing the ML pipeline for the AI model integrated in the system.

## Section 4. Production system

Author: Nishchya

The System Architecture is designed to deploy a web application using Kubernetes Cluster on Google Cloud Platforms. At its core, the application is packaged as a Docker Container. The web Application is run in the form of Docker Images made and deployed online using the GitLab CI/CD pipeline. The CI/CD Pipeline builds the Docker image, pushes it onto the Google Artifact Registry, and then restarts the Kubernetes deployment by triggering a rolling restart of all pods using the updated image without causing downtime upon successful completion of test cases. This ensures that we get automation, consistency, and reliability while deployment.

Presently, we have two model versions trained and available for the admin to use as per their discretion. These two versions are trained with 5000 data-points and 159,000 data-points respectively. The model versions are not stored in the image or locally but are stored on a separate github repository and are pulled dynamically when a particular model is to be used. As well as, when a new model is trained it is pushed to the github repository. This deployment schema can be shown in *figure 4.1*.

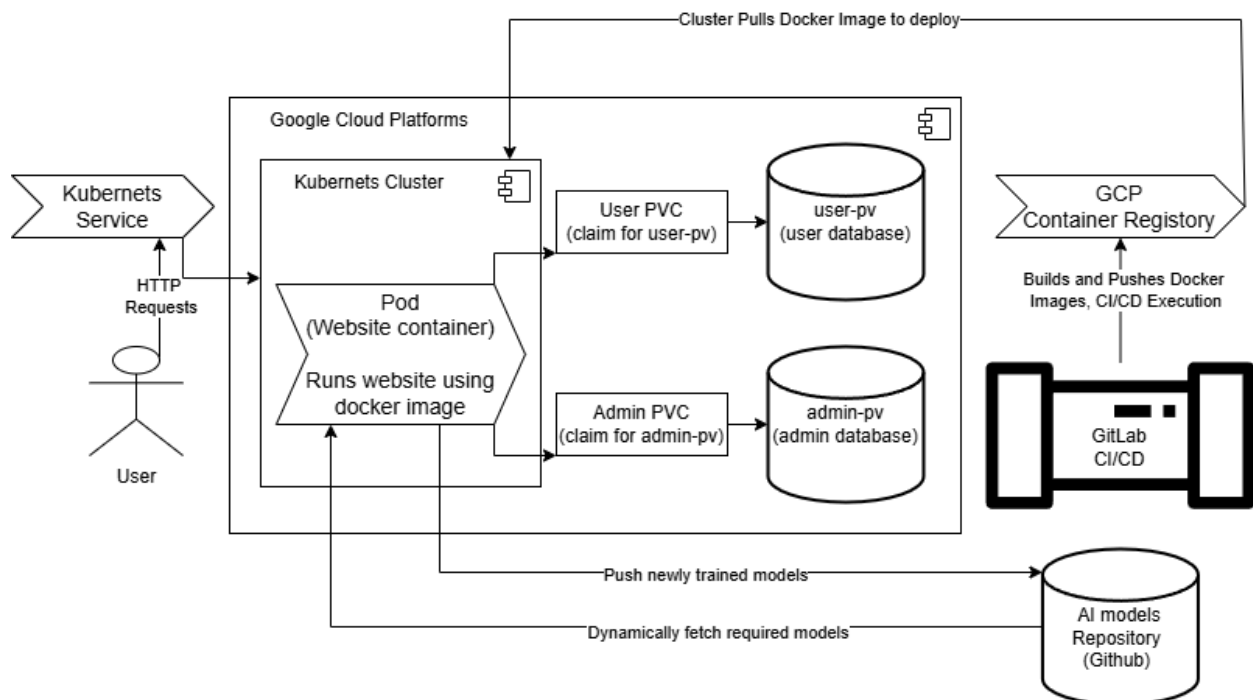
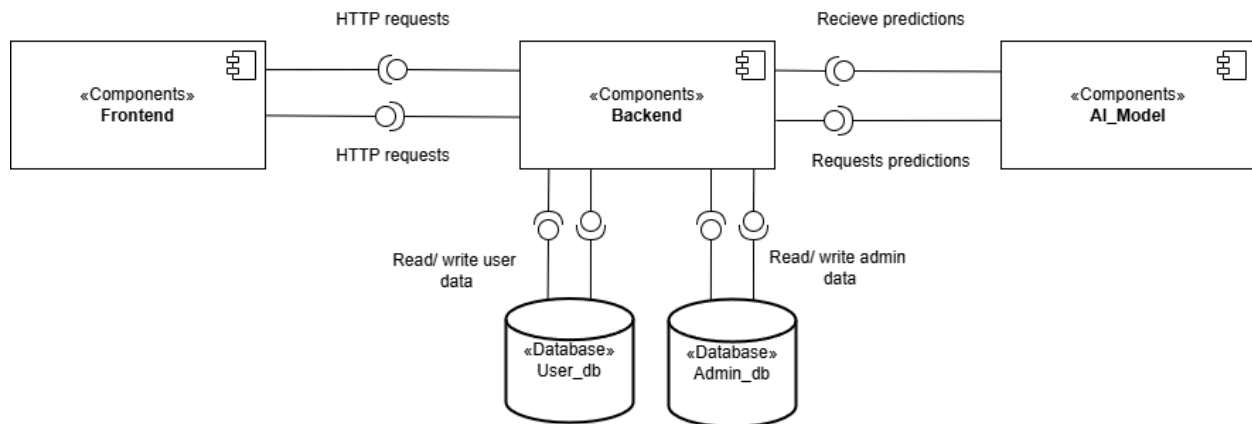


Figure 4.1 : System Deployment Diagram

The Architecture is divided into three main entities - The Frontend, The Backend and the AI Model. The frontend uses HTTP requests to communicate with the backend to process data and get information with the help of two Databases, namely the User Database(default) , and the Admin Database (admin\_db). The user database is used to house user related data, that is the search queries that the user puts into the program. It stores them all in order for us to be able to retrain the model based on this data. The admin database is used to store the credentials of the

system administrators and is only used to authenticate the admin credentials, as shown in *figure 4.2* giving an overview of the system component diagram.

The backend interacts with the AI Model which is responsible for handling the prediction requests and returning search results. The system uses Persistent Volumes (storage disks), namely user-pv (to house user database) and the admin-pv (to house the admin database). This is done to ensure that there is no data loss when Kubernetes Pods are restarted.



*Figure 4.2 : System Component Diagram*

The backend is implemented by a set of APIs, these APIs handle the calls from Frontend, invoke the AI-Model for predictions, and interact with the databases. The AI-Model itself was trained using a pipeline described in section 3, that includes the raw data collection, data cleaning, data processing, training, and evaluation and then finally resulting back in a new versioned model. This trained model is pushed onto a github repository where it can be pulled whenever necessary and is used during runtime to handle prediction tasks.

The Database decisions were straightforward. The decisions to use two databases instead of one was to keep the admin login information inaccessible to the system admins. This also ensures a clean and maintainable schema. The decision to use SQLite was taken as it fulfilled the requirements of our system and was easier to manage and keep. These databases are housed inside the persistent volumes to ensure that there is no data loss even during updating and restarting the Pods.

The GitLab CI/CD pipeline ensures seamless development and deployment. With the CI/CD pipeline we were able to automate the build, test and deployment of different versions of docker images. Once an image is built, it is then tested and only after the test cases pass, it is deployed. For deployment, these images are pushed onto the GCP - Google Artifact Registry and the Kubernetes Pod is restarted in order to deploy this latest version.

The users interact with the system using the frontend where they can submit textual input to analyse and get appropriate results for the 'inappropriate' percentage of the content. Users put

the input and clicks on *analyse text* and this triggers a chain of operations - Frontend sends the data to the backend and requests a prediction from the AI Model. Once the request is processed by the model, the result is sent back to the Frontend.

As for the Admins, they can login into their dashboard through the admin login page following this chain of commands - Admin enters the credentials, the data is sent to the backend, the backend queries the tables to verify the credentials, the admin is then redirected to their dashboard if the credentials are accepted.

The screenshot displays the CyberSafeAI user interface. At the top, a purple header contains the logo and the text 'CyberSafeAI Analyze text for toxic content'. Below this is a text input area where the user has entered 'That is idiotic.' and a character count of '16 characters' is shown. A blue 'Analyze Text' button is positioned to the right of the input. The results section, titled 'Text Analysis', shows the input as 'That is **idiotic.**'. Below this, the 'Analysis Results' section presents six categories with their respective probabilities and toxicity levels:

Category	Probability	Toxicity Level
Toxic	98.2%	Toxic
Severe_toxic	12.5%	Non-toxic
Obscene	85.2%	Toxic
Threat	2.9%	Non-toxic
Insult	72.0%	Toxic
Identity_hate	9.4%	Non-toxic

Figure 4.3 : User Dashboard (as analysed by version 1, trained on 5000 datasets)

The admin dashboard is the workplace for the admins to control the application and directly view and manage the user search history for the entire application. In the admin dashboard, all the user search history results are displayed using a get command to fetch everything from the user database. The admin can see the true-false results of all the attributes for the queries searched by the users of this application along with the date and time for when that search was made. The admins can also sort these results by various different filters, search for specific keywords and find relevant queries, select a few queries and delete them from the user database.

The admin has the functionality to download this user database in the .csv format on their machine, they can upload new data to train the model (again in .csv format) and can initiate the model training process from the front end.

The admin can also switch between different model versions for the application to function with, thereby having the functionality to roll back to a previous version incase of an issue.

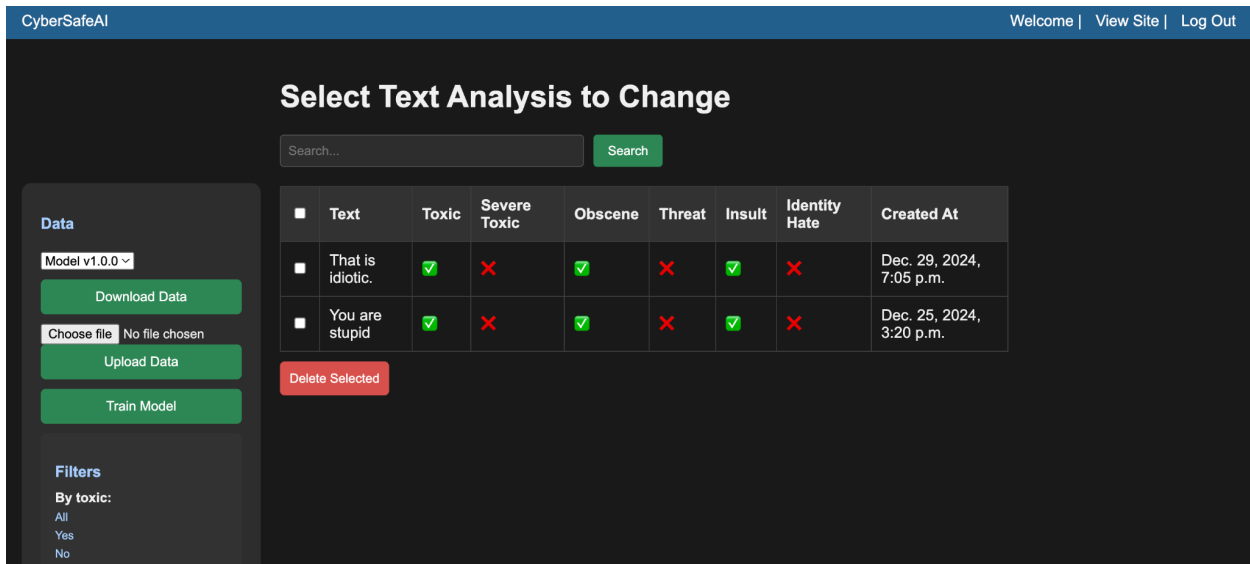


Figure 4.4 : Admin Dashboard

In terms of AI efficiency, the use of BERT-base-uncased allows optimisation to fine tune the model based on our inputs. While testing lighter models such as distilBert, it wasn't reliable though efficient, but our aim was to have a model that balances on accuracy and precision with performance and efficiency. Hence we opted for bert-base-uncased to not have a major impact on computational overhead while retraining the model. Furthermore, each step of the ML pipeline is logged allowing for reproducibility and traceability for debugging. By versioning the model and storing the respective tokenizers, it allows us to change models on the go and rollback to a previous version allowed through our deployment structure allowing easier scalability.

Risk Mitigation is addressed using the fault tolerance and scalability features offered by Kubernetes, ensuring the high availability of the application. The use of CI/CD pipeline to deploy reduces human error while deploying the application as the build and deployment process is automated and tested. The use of persistent volumes in Kubernetes helps prevent data loss by decoupling the storage from the application containers.

Unit testing is essential for ensuring the proper functioning and robustness of the system. We created test cases for models to validate that JSON responses adhered to defined schemas and handled edge cases like invalid inputs and unauthorized access appropriately. Authentication and authorization mechanisms were tested to ensure restricted access was enforced for protected endpoints. For database models, we validated model creation, field-level

validation, and the behavior of custom methods, queriesets, and managers. Django views and endpoints were rigorously tested by simulating HTTP requests using Django's built-in test client. These tests verified that the application behaved as expected across various scenarios, including validating the correct HTTP status codes (e.g., 200 OK for successful requests and 500 Bad Requests for malformed data). The response content was also thoroughly checked to ensure it returned the expected data, such as well-structured JSON responses for APIs or correctly rendered templates for HTML views. These tests were deployed into the CI/CD pipeline allowing issues to be identified and addressed before code is deployed to production.












As a whole, the system is modular and efficient with clear separation of concerns between the frontend, backend and the AI Model. It prioritises scalability, maintainability, availability and fault tolerance using the modern day practices for cloud-based applications. The use of kubernetes and GCP ensures that the system can handle increased loads while CI/CD and logging mechanisms make it easy to maintain and monitor the application.

## Section 5. Software quality

*Author: Raghav, Nishchya*

### Code structuring:

The code preserved the dependencies provided by Django but remodeled the project directory to improve clarity, ease of understanding and the distribution of files by relevancy into respective folders. The code structure is divided into 3 main folders: Frontend, Backend and the AI-Model while all the configuration files(Dockerfile, .gitlab-ci.yml etc) stay on the root of the project along with the README.md.

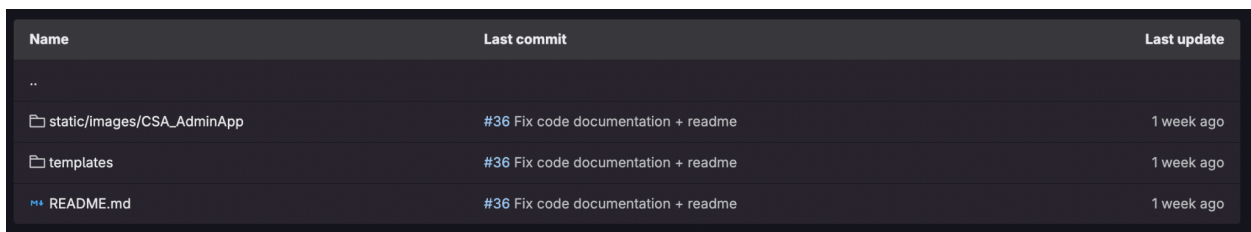
Name	Last commit	Last update
 ai_model	Fix the training process <a href="#">#24</a> <a href="#">#16</a>	2 weeks ago
 backend	<a href="#">#39</a> Fix code for highlighting of words f...	4 days ago
 frontend	<a href="#">#36</a> Fix code documentation + readme	1 week ago
 .dockerignore	<a href="#">#29</a> - Preventing local databases to be ...	2 weeks ago
 .gitignore	Add the data <a href="#">#16</a>	2 weeks ago
 .gitlab-ci.yml	<a href="#">#38</a> - CI/CD Fix	1 week ago
 Dockerfile	Merge/latest app v1	2 weeks ago
 README.md	Fix readme	2 weeks ago
 deployment.yaml	<a href="#">#38</a> - Code Documentation for deploy...	1 week ago
 service.yaml	<a href="#">#38</a> - Code Documentation for deploy...	1 week ago
 sqlite-pv-pvc.yaml	<a href="#">#38</a> - Code Documentation for deploy...	1 week ago



*Figure 5.1 : Overview of the project directory*

The frontend folder is divided into 2 subfolders - Templates and Static Images and focuses on the CSA\_AdminApp and the moderation Django applications. The files in the frontend are divided like:

- **Frontend:**
  - **Templates** - It houses the HTML files that house the code to render the frontend and the respective endpoints connecting to the backend logic.
  - **Static Images** - It houses the CSS code and the images used to style the web pages of the application.



Name	Last commit	Last update
..		
static/images/CSA_AdminApp	#36 Fix code documentation + readme	1 week ago
templates	#36 Fix code documentation + readme	1 week ago
README.md	#36 Fix code documentation + readme	1 week ago

*Figure 5.2 : Overview of the frontend folder*

The backend folder is divided into multiple subfolders - CSA\_AdminApp, app, config, data, models, and tests while also housing the . It focuses on handling the backend logic and API endpoints for the CSA\_AdminApp and moderation Django applications. The files in the backend are divided as follows:

- **Backend:**
  - **CSA\_AdminApp**: Contains application-specific logic related to the AdminApp of the system. This folder houses the files relevant to the admin's functionality and a database router that routes the traffic to respective databases as required. This folder also houses a migrations folder that has just the '\_\_init\_\_.py' file to make migrations.
  - **app**: Houses the core backend functionality, including views, models, urls, apps and admin files that support the applications. This folder also houses a migrations folder that has just the '\_\_init\_\_.py' file to make migrations.
  - **config**: Stores configuration files and project-level settings required to initialize and run the backend. Config houses files like asgi, config, logger, model\_manager, **settings**, urls, wsgi relevant to the application's configurations. This folder also has another folder with the name of 'static files' that houses the

static files used by the application.

- **data:** The data folder is further divided into two subfolders - raw and processed. These folders contain the raw data of the application (original dataset) and processed data(after cleaning) used for the training and testing purposes of the application's model.
- **models:** This folder is used for storing the different versions of the model. Each model is stored in its respective folder in the directory. Another folder used to have the checkpoint for data training is also present here.
- **tests:** Includes unit test cases to ensure the reliability and correctness of the application's functionality.

Additionally, the backend folder includes a 'manage.py' file for command-line interactions with the Django framework and the local databases and 'requirements.txt' file to list all dependencies required for the backend environment. These dependencies are installed every time before the application is deployed.

The backend folder also houses two local databases that we used for testing the application locally. These databases are called - db.sqlite3 and the admin\_db.sqlite3 respectively.

Name	Last commit	Last update
..		
CSA_AdminApp	#37 - Backend Code Documentation	1 week ago
app	#39 Fix code for highlighting of words for all 6 labels	4 days ago
config	#37 - Backend Code Documentation	1 week ago
data	Add the data #16	2 weeks ago
models	Train the v3 #24	2 weeks ago
tests	#37 - Backend Code Documentation	1 week ago
README.md	Set project set up	1 month ago
__init__.py	#19 fix moderation app routing	3 weeks ago
manage.py	#37 - Backend Code Documentation	1 week ago
requirements.txt	Merge branch 'ai_model/generate-test-cases' into aiBranches/highlightWords	2 weeks ago

*Figure 5.3 : Overview of the backend folder*

The AI\_model folder consists of all the aspects of the ML pipeline and functions to retrain and fine-tune the new models. This is done through the following structure:

- **AI Model:**
  - **test:** The folder includes a test\_preprocessing.py file to see if data is being loaded correctly or not.

- **utils**: Contains the code for each of the steps of the ML pipeline from combining data, loading and preprocessing data, training the model, evaluating it and saving it versionised.

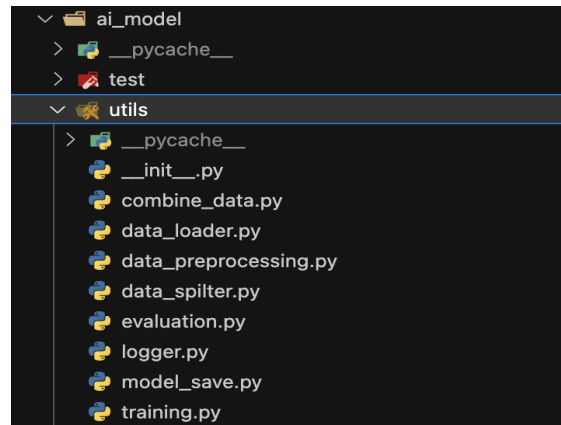
As well as, the root of the folder consists of train.py, which acts as the main file where all the steps for training take place in a chronological order. While config.py consists of all the configurations such as batch size, learning rate, etc. required for the training.

Name	Last commit	Last update
..		
test	Refactor the ai model	3 weeks ago
utils	Fix the training process #24 #16	2 weeks ago
README.md	Set project set up	1 month ago
config.py	Refactor the ai model	3 weeks ago
requirements.txt	Implement data preprocessing module	1 month ago
train.py	Fix the training process #24 #16	2 weeks ago

*Figure 5.4 An overview of the AI Model folder*

Our code utilizes the **Single Responsibility Principle**, ensuring that each function or file was performing a single task. This approach allowed us to reuse functions across the project during runtime by calling them as needed. By adhering to this structure, we also maintained **DRY (Don't Repeat Yourself)** principles, as each function was implemented only once and reused whenever required. This not only reduced redundancy but also improved code clarity, maintainability, and scalability.

An example of the same is the design of the 'ai\_model/utils' directory. This folder was structured to contain files, each serving a specific purpose related to managing the AI model.



*Figure 5.5 showing the contents of the ai\_model/utils directory*

The files within the directory were named according to the purpose they serve, ensuring that team members who didn't work on a particular section of the project were still able to interpret and use the code. For example, the 'data\_splitter.py' file contains functions used for splitting data into training and testing sets. It processes the input dataset and returns CSV files, strictly focusing on this task without handling unrelated logic. This can also be considered as an application of DRY code since data splitting can now be accomplished by calling the splitter function.

### **Code documentation:**

All the project files have been commented out with explanations of each step and relevant functions. This makes the code easy to understand for the readers and us developers as we ourselves are constantly learning. This allows us to easily find relevant functions and contribute effectively throughout the development process.

The project houses a README on the project's root, and in the three main folders : backend, frontend and the AI Model. The README lists all the names of files and provides an explanation for each file and the purpose they carry. This approach makes it easier for the developers to understand the division of roles between files. This enhances collaboration and also maintains the overall readability and maintainability of the codebase.

```

## Backend Folder

The backend folder contains the main Django app, including the models, views, URL routing, and business logic. It is responsible for handling HTTP requests, serving APIs, managing user data, and interacting with the AI model for content analysis.

### Contents:
- app: The main Django application that includes:
  - models: Django models for database tables.
  - views: Views that handle user requests and render templates.
  - urls: URL configuration for routing URLs to corresponding views.
  - serializers: Django Rest Framework serializers for API requests and responses.
  - permissions: Custom permissions classes for API endpoints.
  - tests: Unit tests for backend logic.
  - utils: Utility functions to handle repetitive tasks.

- config: Configuration files for the Django project.
  - settings.py: The main settings file that configures the project (e.g., database, security, middleware).
  - urls.py: The main URL routing configuration.
  - wsgi.py: Entry point for deploying the Django application in production.

- requirements.txt: Backend dependencies such as Django, Django Rest Framework, etc.
- manage.py: Django management script for running server, migrations, etc.

### Purpose:
This is the heart of the application, where the logic for content submission, moderation, and analysis happens. It handles user interactions, communicates with the database, and serves AI-powered analysis results.

```

Figure 5.6 the README file in the backend directory

## Section 6. Reflections

Author - Utkarsh

### 6.1 Challenges Encountered

One of our biggest challenges was dealing with the *severe imbalance* in our dataset, where categories like *threat* and *identity\_hate* were underrepresented compared to more common labels such as *toxic*. This skew often caused the model to overfit to the majority label, hindering its ability to accurately learn less frequent categories.

To enhance *accuracy, precision, and recall*, we experimented with different data-management approaches:

- **Retaining the entire dataset** (over 100,000 entries) to preserve maximum diversity, ensuring the model saw a wide range of real-world text.
- **Randomizing the dataset**, which distributed various labels throughout training batches and prevented any single label from dominating.
- **Balancing subsets** so minority classes were represented equally, though this sometimes reduced overall performance by underrepresenting the naturally dominant *toxic* label.

Through experimentation, we found that using the *complete dataset* with *strategic randomization* provided the best trade-off among all key metrics, as it maintained the natural data distribution. We also performed *thorough data cleaning*—removing extraneous symbols, noise, and irrelevant content—to refine training signals. While balancing techniques helped draw attention to rare categories, they often diminished the model’s broader performance. Ultimately, preserving the full dataset proved most effective, allowing the model to capture real-world complexities without compromising accuracy.

We decided to use BERT as the base model to fine-tune since it strikes a better balance of providing high performance while requiring comparatively less computational power when compared with the newer versions, such as RoBERTa or DeBERTa. However, BERT can be resource-consuming if it has to process a lot of queries at once, so we need to be very attentive choosing batch sizes, the length of sequences, etc. Perhaps we could have taken a look at other pre-trained models (such as ALBERT or a light multilingual BERT) or a mixture strategy (using a small model for fast response, yet a large one for accurate responses).

## 6.2 What We Could Have Done Differently

### 1. Earlier Start on the Final Report and Documentation

One of the biggest lessons learned is the importance of early and continuous documentation. Starting the final report—and all related documentation—sooner would have allowed us to capture decisions, experiments, and results in *real time*, rather than rushing to compile them at the end.

### 2. Early Deployment and Incremental Testing

We learned that delegating deployment logistics to the end of the project introduced avoidable risks. Had a minimal viable version been tested earlier, integration bugs and environment mismatches might have been identified and eliminated way ahead of time before causing a lot of stress and many last-ditch fixes during the last phase.

### 3. More Thorough Exploration of Model Architectures

Although BERT offered a good balance between speed and accuracy, we could have set aside more time to experiment with other, potentially more suitable transformers or smaller language models. Automated hyperparameter tuning (e.g., with tools like Optuna) might have provided better insights into optimal training setups, especially for imbalanced datasets.

## 6.3 AI ETHICS: Our Ethical Considerations

### 1. Privacy and Data Protection

Given that CyberSafeAI works with user-generated content, some of which might be of a sensitive nature, we decided not to store user text in a manner that would allow for identification of individual users. It is always important to respect data protection legislation hence wherever possible we eliminated identifiers from our data. Secondly, we did not gather any personal identification information in order to provide all users with a safer environment.

## 2. **Bias and Fairness**

Toxicity detection systems can unintentionally reflect biases present in the training data. If certain demographics or language styles are overrepresented or underrepresented in the dataset, the model might misclassify or unfairly flag content. Mitigating bias requires ongoing vigilance—collecting diverse data, continuously retraining, and validating outcomes against demographic subgroups.

## 3. **Use of AI for a Safer Environment**

Our main focus is to help users detect and filter potentially abusive or insulting words. We do not implement dynamic blocking or filtering and recommend banning or censoring, the decision remains with the users or moderators. This stance is an attempt at finding a middle ground where people enjoy their rights to fully express themselves on the social media as well as learn manners on how to conduct themselves on the same.

## 6.4 Possible Extensions

### 1. **Reading Text from Images (OCR)**

Most of the toxic or hateful messages could be screenshot or a meme. The incorporation of Optical Character Recognition (OCR) built into the platform would enable CyberSafeAI to detect text incorporated in images opening up a new front in the content moderation services.

### 2. **Paraphrasing and Automated Text Suggestion**

One potential feature for continual integration could be a paraphrase feature to keep an eye on and rewrite or dilute particular distressing words or expressions. Besides flagging the content as toxic this feature would suggest to the users less toxic words and at the same time defend the freedom of speech as long as it is not toxic.

### 3. **Multilingual Support**

It is clear that expanding the possibilities of applying CyberSafeAI for other languages or dialects could significantly extend the possibility of its application and become extremely valuable. Maybe using multilingual or language-specific models (e.g., XLM-R, mBERT) could really regulate content from different language-using communities.

### 4. **Integrated Everyday Professional Development and Feedback Demands**

The addition of an online learning strategy or a feedback loop mechanism such as – ‘Was this classification correct? Could help update model results successively. In the future, such feedback would contribute to the model improving its performance gradually, by learning new trends in the language, new idioms, slangs and different cultures.

### 5. **Advanced Explainability Features**

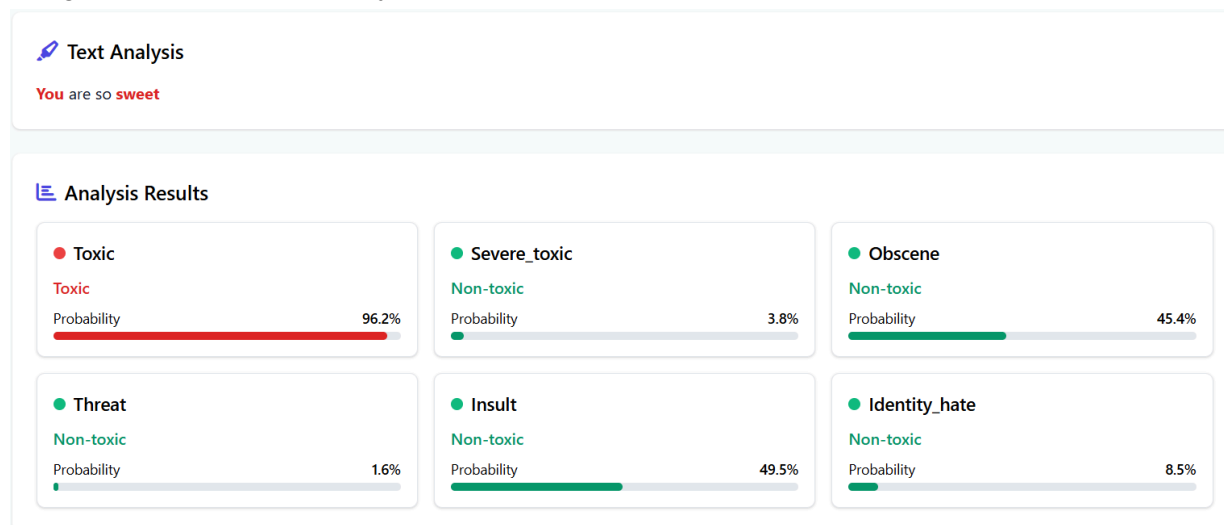
Beyond simple text highlighting, advanced explainable AI methods like integrated gradients or local interpretable model-agnostic explanations (LIME) can reveal *why* the model considers certain phrases toxic. Such transparency increases user trust and helps them make more informed decisions about their content.

## Section 7. Advanced AI functionality

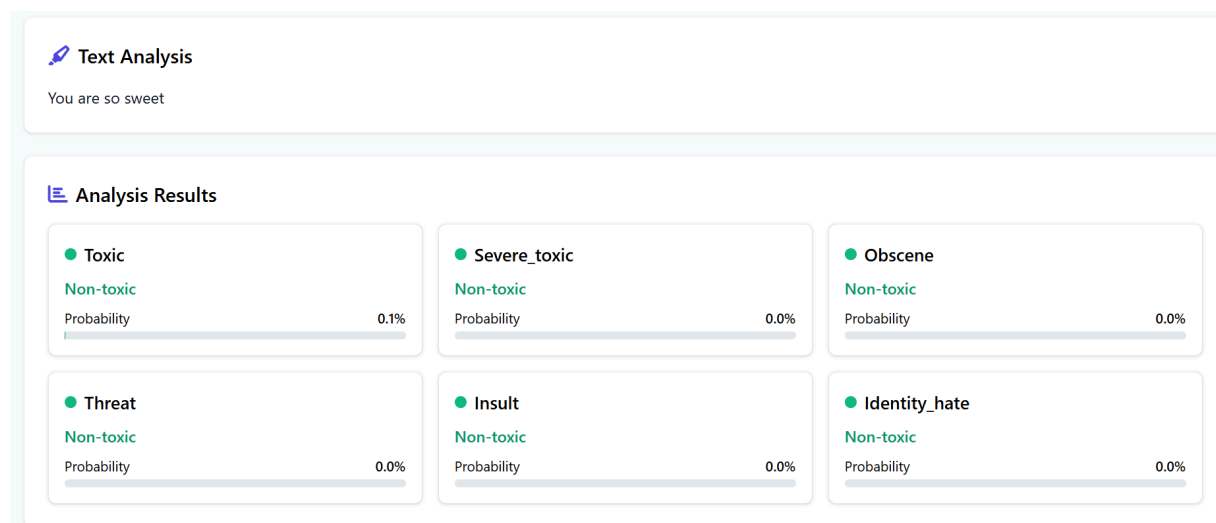
*Author: Jitish*

The project includes numerous advanced AI functionalities that match with the assignment description, these include, use of Explainable AI, use of NLP, transfer learning, and transformers, and natural language data processing for raw textual data.

Firstly, we incorporated the use of Explainable AI through the highlighting of the input text that contributes to the 6 categories. This is done through each word being individually analyzed due to being tokens and those that surpass the threshold of 0.5 (50% of that specific category) are flagged and displayed in red to the user (see *figure 7.1* and *figure 7.2*). The following method allows transparency and clarity for the reason of it being the various toxicity categories and an opportunity for the user to be responsible with certain choice of word.



*Figure 7.1. Version 1 of AI model (trained on 5000 data points)*



*Figure 7.2. Version 2 of AI model (trained on 159,000 data points)*

Furthermore, the use of NLP techniques in hand with AI methods of transfer learning and transformer networks accomplished another step of advanced AI functionalities. The use of a pre-trained BERT model, allows use of tokenization to convert the input and apply data



processing such as truncation and padding to have efficient batch processing. BERT's transformer allows contextual relationships to have a better understanding of the nuances of natural language. As well as, transfer learning allows us to fine tune and retrain the previous BERT model for multi-label classification with new data reducing the training time. The following points showcase the integration of cutting-edge AI tools and AI principles to handle and understand toxicity.

## Appendices

### Appendix I – Team activity (to be filled in for Assignment 1 & updated in Assignment 2)

List the meetings the team had, and their attendance by team members.

Attended meetings	Assignment 1 Out of 6	Assignment 2 Out of 8	Notes (if any)
Jitish Rajankumar Padhya	6	8	
Muhammed Ali Elhasan	6	8	
Nishchya Arya	6	8	
Raghav Tengse	6	8	
Utkarsh Singh	6	8	

### Appendix II - Responsibilities (to be filled in for Assignment 1 & updated in Assignment 2)

Per week, describe who did what (main tasks). This information should be available on your Markdown planning already. In particular, include a table like the following one:

<b>Week 1</b>
---------------

<b>Task or feature description</b>	<b>Who was responsible</b>	<b>Delivered</b>	<b>Integrated</b>	<b>Notes (e.g. reason for <u>not</u> having delivered or integrated)</b>
<b>Idea Brainstorming for the Project:</b> Each member was expected to come up with a project idea and a suitable dataset for the same in the coming meeting.	All	Yes	Yes	
<b>Deciding upon three project ideas to present for the pitch :</b> The group held discussions about each idea that was come up with and decided upon the three most suitable options for the pitch	All	Yes	Yes	
<b>Schedule Meeting, Communication Platform</b>	Utkarsh	Yes	Yes	
<b>Week 2</b>				
<b>Design Presentation Slides</b>	All	Yes	Yes	
<b>Choose Project idea after feedback</b>	All	Yes	Yes	
<b>Give System Description and the goals and objectives</b>	Jitish	Yes	Yes	

<b>of the project:</b> part of section 1				
<b>Description of the database:</b> section 2	Jitish	Yes	Yes	
<b>Setup Environment (Download Anaconda and Django, Repository on respective local machines</b>	All	Yes	Yes	
<b>Write System Requirements (User, Admin, Non-functional requirements)</b>	Nishchya, Utkarsh	Yes	Yes	
<b>Write User Stories (along with acceptance criteria)</b>	Raghav	Yes	Yes	
<b>Create the initial data and toy prediction model. (to be put on git)</b>	Ali	Yes	Yes	
<b>Week 3</b>				
<b>Reflect on the Assignment Feedback</b>	All	Yes	Yes	
<b>Project structure and create the project issues</b>	Ali	Yes	Yes	
<b>Create frontend project components for user side in ToyModel</b>	Jitish	Yes	Yes	

<b>Add ability to filter by toxicity and clean results in frontend, also fix percentage rounding and decimal place bug</b>	Jitish	Yes	Yes	
<b>Admin dashboard (backup, upload, Download collected data, system activity)</b>	Utkarsh, Nishchya	In progress	No	
<b>Creating a user interface for admin log- in</b>	Raghav	Yes	Yes	
<b>Creating a database to store admin credentials in order to authenticate admin access</b>	Raghav	Yes	Yes	

<b>Week 4</b>				
<b>Implement the AI model</b>	M.Ali	Yes	No	
<b>Implement the utils of the ai model “data_splitter, data_loader, data_preprocessor, model_training, model_evaluation, save_model and training process ”</b>	M.Ali	Yes	Yes	

<b>Implement the endpoints of the admin and user to interact with the AI model</b>	M.Ali	Yes	Yes	
<b>Create the Usage History endpoint and save it to the db and structure for the re-training process</b>	M.Ali	Yes	Yes	
<b>Refactor and bug fixes moderation app:</b> splitting the app into directories of frontend, backend, ai_model	Jitish, Nishchya	Yes	Yes	
<b>Refactor and bug fixes CSA_AdminApp:</b> splitting the app into directories of frontend, backend, ai_model	Jitish, Nishchya	Yes	Yes	
<b>Restructure both the apps (CSA_AdminApp and Moderation app):</b> consisting of rerouting of urls, files, etc.	Jitish, Nishchya	Yes	Yes	
<b>Admin dashboard (backup, upload, Download collected data, system activity)</b>	Utkarsh, Nishchya	Yes	Yes	
<b>Implement the AI utils, for the combine the new data with the raw data before training</b>	M.Ali	YES	No	
<b>Week 5</b>				

<b>Dockerfile - Initial commit</b>	Nishchya	Yes	Yes	
<b>Initial gitlab CI:</b> implementation to build docker image and store in container registry on gitlab	Jitish	Yes	Yes	
<b>Research deployment using GCP and Kubernetes</b>	Nishchya, Jitish	Yes	Yes	
<b>Kubernetes: deployment.yaml and service.yaml</b>	Nishchya	Yes	Yes	
<b>Research data storage using PV, PC in pods</b>	Nishchya, Jitish	Yes	Yes	
<b>Refactor the training process and connect it to the backend</b>	M.Ali	Yes	Yes	
<b>Implement save model utils in the training process</b>	M.Ali	Yes	Yes	
<b>Week 6</b>				
<b>Persistent volumes to store databases on cloud</b>	Nishchya	Yes	Yes	
<b>Remove toyModel and its dependencies:</b> For decreasing storage usage and inconvenience	Jitish	Yes	Yes	

<b>Add test cases for database models and for frontend endpoints</b>	Raghav	Yes	Yes	
<b>Refactor dockerfile to remove toymodel and databases</b>	Nishchya	Yes	Yes	
<b>Add unit tests to CI pipeline and minor docker fixes regarding build</b>	Jitish	Yes	Yes	
<b>Use LIME for text analysis and highlighting words: Explainable AI</b>	Jitish	Yes	Yes	
<b>Add all 6 labels to be stored in the database</b>	Utkarsh	Yes	Yes	
<b>Display all 6 labels for user and admin dashboard</b>	Utkarsh	Yes	Yes	
<b>Highlight word for explainable AI</b>	Utkarsh	Yes	Yes	
<b>Fix deployment issues and include it in the ci/cd pipeline</b>	Nishchya	Yes	Yes	
<b>Fix CI/CD pipeline for testing and deployment + minor routing fixes</b>	Jitish	Yes	Yes	
<b>Refactor the AI model connectivity with the backend, and implement new</b>	M.Ali	Yes	Yes	

endpoint to track the process and versioning in the Frontend and backend				
Deployment, component, and ML pipeline diagram	Jitish, Nishchya	Yes	Yes	
Train the AI model v3	M.Ali	Yes	Yes	
<b>Week 7</b>				
Code documentation + readme for frontend	Jitish	Yes	Yes	
Code documentation - Backend	Nishchya	Yes	Yes	
Code Documentation - CI/CD, Dockerfile, Deployment.yaml	Nishchya	Yes	Yes	
Refactor the deployment server to have new structure to handle the versioning and upload data process	M.Ali	No	No	
Section 3: Final report	Jitish	Yes	Yes	
AI models “V1 & V2” re-train with performance metrics, to be added to the final report	M.Ali	Yes	Yes	
Refactor the endpoints and services in the	M.Ali	Yes	Yes	



admin dashboard and the user analyze text				
<b>Week 8</b>				
<b>Section 7: Final report</b>	Jitish	Yes	Yes	
<b>Section 4 - Final report</b>	Nishchya	Yes	Yes	
<b>Section 5 - Final report</b>	Raghav, Nishchya	Yes	Yes	
<b>Section 6 - Final report</b>	Utkarsh	Yes	Yes	
<b>Fix highlighting word bug for all 6 labels:</b> explainable AI	Jitish	Yes	Yes	
<b>Make a separate repo to save the model in, to have efficient locally and server training and simulate them in both sides</b>	M.Ali	Yes	Yes	
<b>Update the deployment process to match with the AI models changes</b>	M.Ali	Yes	Yes	
<b>Deploy latest version of project</b>	M.Ali	Yes	Yes	
<b>Main and AI_model directory READMEs</b>	Jitish	Yes	Yes	
<b>Backend README</b>	Nishchya	Yes	Yes	
<b>Review the final report and add the individual contribution</b>	All	Yes	Yes	

