Master Project

A logo for a university

Description automatically generated

Computer Science Department

California State University, Dominguez Hills

ChatGPT Proactive Protection Against Malicious

Engagements

Project

Presented

To the Faculty of CSUDH

-----------

In Partial Fulfillment

Of the requirements for the Degree

Master of Science

In

Computer Science

By

Rahul Vattigunta

Spring 2024

Signature Page

The Project of Rahul Vattigunta is approved by:

Dr.

Faculty Advisor -----------------------

Dr.

Committee Member ----------------------

Mr.

Committee Member -----------------------

ACKNOWLEDGEMENT

**TABLE OF CONTENTS**

1. **Introduction Page 7**
2. **Background and Case Study Page 10**
3. **Methodology Page 13**
4. **Implementation Page 17**
5. **Future Work Page 28**
6. **Conclusion Page 31**
7. **References Page 32**

**LIST OF FIGURES**

**Fig.1: Code Snippet load\_toxic Func Page 22**

**Fig.2: Code Snippet Toxicity Check Page 22**

**Fig. 3: Code Snippet Web Page 23**

**Fig. 4: Code Snippet Main Route Page 24**

**Chapter 1: Introduction**

Online communication platforms, such as forums, social media, and chat programs, have become essential in daily interactions in the digital era. Although these platforms enable connectivity and the sharing of information, they also pose difficulties in regulating material to prevent detrimental interactions. Detrimental interactions, such as harassment, hate speech, and other types of abusive language, can have a substantial effect on the well-being of users and the general vitality of onlinecommunities.   
  
In order to tackle these obstacles, the objective of this research is to create a system that can identify and minimize detrimental interactions in chat settings in real-time. Using sophisticated Natural Language Processing (NLP) methods, particularly transformer-based models such as GPT and BERT, the system will examine incoming communications and accurately detect potentially hazardous information.  
  
  
The study starts by gathering a heterogeneous dataset of dialogues that encompass both innocuous and detrimental exchanges from a range of sources, such as chat records and social media sites. The data is annotated to identify the presence of detrimental material, which forms the basis for training our model. The gathered data is subjected to cleansing and standardization processes to confirm its suitability for training the model. This include the elimination of extraneous information, management of missing values, segmentation of text into tokens, and ensuring a balanced dataset to mitigate bias.   
  
A transformer-based natural language processing (NLP) model is selected and optimized using the preprocessed dataset. This stage entails instructing the model to comprehend the subtleties of detrimental interactions and precisely categorize them. After training, the model is incorporated into a chat interface that operates in real-time. This enables ongoing analysis of messages and immediate actions such as notifying moderators or restricting users when dangerous information is detected.   
  
The system's efficacy is assessed through the utilization of diverse criteria, including accuracy, recall, and F1-score. This assessment aids in enhancing the model and enhancing its capacity to identify detrimental interactions. During the project, we actively consider ethical concerns such as safeguarding data privacy, obtaining user permission, and mitigating prejudice. This is done to guarantee that the system is fair, transparent, and respects the rights of the users.   
  
Through the development of this system, our objective is to improve the security and reliability of online platforms, creating a more inclusive and friendly environment for all users. This initiative encompasses not just a complex technological endeavor but also a dedication to enhancing digital interactions and cultivating more wholesome online communities.

**Chapter 2: Background and Case Study**

The advent of digital communication channels has fundamentally transformed the manner in which individuals engage with one another, facilitating immediate and worldwide relationships. Nevertheless, the growing interconnectivity also presents notable difficulties, especially when it comes to regulating detrimental content. The frequency of online harassment, cyberbullying, hate speech, and other unpleasant interactions has been on the rise, impacting a significant number of people globally. These detrimental practices can result in significant psychological consequences, such as anxiety, sadness, and reduced engagement in online networks.   
  
Conventional moderation techniques, such as human moderators doing manual reviews, cannot handle the large number of interactions on major platforms because to their lack of scalability. Moreover, these techniques might exhibit inconsistency and susceptibility to prejudice. The demand for automatic, instantaneous answers is clear, and recent progress in Natural Language Processing (NLP), Artificial Intelligence (AI), and Neural Networks provide a hopeful approach to tackling these problems.   
  
Notable accomplishments and progress have been achieved in the domain of Natural Language Processing (NLP), Artificial Intelligence (AI), and Neural Networks. One of the significant advancements in recent times is the rise of language models such as GPT-2 and GPT-3, which have had a profound impact. These models have exhibited exceptional ability in many natural language processing (NLP) tasks, including question-answering, text production, and language comprehension. Although these language models have achieved success, they are subject to various constraints such as biases and restricted accuracy, which might diminish their efficacy in practical scenarios.   
  
In order to overcome these constraints, scientists have created interactive language models like ChatGPT. ChatGPT is an optimized iteration of GPT-2 and GPT-3, specifically engineered to enhance user interactions and address some biases and inconsistencies found in its earlier versions. This technological progress has played a vital role in facilitating a more dependable and sophisticated comprehension and production of written language, which is necessary for identifying hazardous interactions in online communications.   
  
Transformer-based models, such GPT and BERT, have demonstrated exceptional efficacy in comprehending and producing text that resembles human language. This makes them well-suited for identifying the intricacies and circumstances of harmful interactions. Their capacity to adjust to various languages and circumstances by refining their skills makes them exceptionally efficient for this objective. The utilization of these models in real-time systems for identifying detrimental interactions is evidence of their adaptability and efficacy.   
  
By utilizing these sophisticated NLP tools, platforms may improve the security and authenticity of online communities, creating a more inclusive and encouraging environment for all members. The creation and implementation of these solutions not only demonstrate a technological accomplishment, but also a dedication to enhancing digital interactions and promoting more positive online environments.

**Chapter 3: Methodology**

The system is designed to detect harmful interactions in user-submitted text by comparing it against a list of predefined toxic phrases stored in a Word document. This approach allows for easy updating and customization of the criteria for identifying toxicity. The key components of the system include a Flask web application, which serves as the backend and interface logic, and a user-friendly web interface developed with Bootstrap for a responsive and intuitive experience. Environment Setup Development Environment The development environment for this project includes Python as the primary programming language, Flask as the web framework to handle HTTP requests and render the user interface, and python-docx for reading and processing Word documents. This setup allows for robust, scalable application development that can handle text processing efficiently.

Dependencies Installation To set up the project, the following Python libraries need to be installed:

Flask: This is used to create and manage the web server. Install it via pip with pip install flask.

python-docx: This library is essential for reading Word documents where the toxic phrases are stored. It can be installed using pip install python-docx. Loading Toxic Phrases Document Structure The Word document containing the toxic phrases is structured such that each paragraph contains one phrase. This design simplifies the extraction process, as each paragraph directly represents a single toxic entry, making the system flexible in handling varying lengths and formats of phrases. Phrase Loading Function The load\_toxic\_phrases function is critical for initializing the application with the necessary data. It opens the specified Word document, iterates through all paragraphs, and appends each non-empty, trimmed phrase to a list.

This list is then used across the application to check submitted texts for toxicity. Web Application Development Flask Application Setup The Flask application is configured with routes and global settings in the main application file. Key routes include the home route for rendering the main interface and form, and a route for processing user inputs, either to load new toxic phrases or to check text toxicity. User Input Handling Loading Phrases: This action allows users to dynamically update the list of toxic phrases by specifying a new Word document path. The system reads and updates its internal list based on this input. Toxicity Check: Once toxic phrases are loaded, users can input any text to see if it matches the criteria for toxicity based on the loaded phrases. Error Handling The system robustly handles errors, including incorrect file paths, unreadable files, or empty inputs. These errors trigger user-friendly feedback to help users correct their inputs without causing system crashes or unhandled exceptions. User Interface Design Form Design The HTML form on the main page has two primary input fields: one for the file path to load new toxic phrases and another for the text to be checked. The form is styled with Bootstrap, ensuring a modern look and feel, responsive layout, and accessibility features. Feedback Mechanism Feedback to the user is implemented using Bootstrap alerts. If the text is identified as toxic, a red alert is shown; otherwise, a green alert confirms the text is clean. Errors during operations like file loading or text processing also trigger alerts, providing immediate, clear guidance to users. Testing and Validation Test Cases Test cases cover several scenarios: Successful Phrase Loading: Validates that the system correctly loads phrases from a correctly formatted Word document.

Toxicity Detection: Checks that the system accurately identifies toxic and non-toxic text based on the loaded phrases.

Error Handling: Ensures that the system gracefully handles and reports errors related to file access and text processing. User Interaction Testing To ensure the interface is user-friendly and the system responds as expected, user testing involves participants trying to load phrase files and input various texts.

Feedback from these sessions is used to refine the UI and improve error messages and overall system responsiveness. By documenting these methodologies, the project report provides a comprehensive blueprint of how the system operates, ensuring clarity and reproducibility for future enhancements or evaluations.

**Chapter 4: Implementation**

The system's architecture is specifically engineered to rapidly and accurately identify detrimental interactions in text given by users. This is achieved by utilizing pre-established hazardous words that are kept in a Word document. This design, consisting of several components, guarantees a strong, adaptable, and user-focused solution.   
  
Backend processing refers to the computational tasks and operations that occur on the server side of a software application. It involves the handling and manipulation of data, as well as the execution of complex algorithms and calculations   
The backend processing is the central component where the primary logic of the application is located. It is composed of many essential components:   
  
The program used the python-docx package to extract information from a Word document chosen by the user. It is believed that each paragraph in the paper has a single hazardous phrase. The phrases are extracted, converted to lowercase, and saved in a list. This technique enables the dynamic modification of the criteria utilized to assess text toxicity, hence allowing the program to be flexible and suitable for various circumstances or specifications.   
  
Toxicity Assessment: After loading the poisonous phrases, the system compares any text supplied by the user with these words. The evaluation technique entails systematically examining the text to identify and detect the occurrence of any harmful terms. If a match is detected, the text is marked as poisonous; otherwise, it is classified as non-toxic. This procedure is designed to be case-insensitive in order to provide consistent. detection across different input types.   
  
Business Logic: The backend is responsible for managing the routing and processing of user requests. This encompasses making decisions depending on user input, such as modifying the roster of offensive expressions or assessing the level of offensiveness in a text excerpt. It effectively oversees user sessions and requests, guaranteeing that each contact is handled accurately.   
  
Graphical user interface for the front-end of a software application.   
The frontend interface is intentionally designed to be user-friendly and easily accessible, guaranteeing that individuals with varying degrees of technical expertise may successfully engage with the system.   
  
User Interaction Design: Created using HTML and enhanced with Bootstrap, the interface incorporates forms for user input and designated sections for presenting outcomes. Bootstrap's responsive design guarantees the application's usability on many platforms, including computers and mobile phones.   
  
Real-time Feedback: Users receive instant feedback on their activities as they engage with the system. When a new Word document is populated with harmful expressions, the system verifies the successful update or notifies of any issues, such as file not found or read failures. Similarly, when evaluating the toxicity of text, the findings are immediately presented in an alert box, with different colors used to differentiate between hazardous and non-toxic conclusions.   
  
The web interface prioritizes accessibility and usability by incorporating clear labeling, enough color contrast, and logical navigation. User experience principles dictate the arrangement and patterns of interaction to reduce user mistakes and optimize the process of detecting text toxicity.   
  
Integration of databases   
By including a database, the system has the ability to store data over a long period of time, enabling a more thorough examination of information as it evolves.   
  
The system utilizes Flask-SQLAlchemy to store data in a SQLite database, which is a lightweight and efficient solution that is well-suited for the scope of this application. This configuration enables efficient data retrieval and processing without the additional burden of intricate database systems.   
  
Data Model: The central element in the database is the Message, which signifies every text fragment given by users. The characteristics of this item consist of the textual content, a boolean indicator for toxicity, and a timestamp denoting the processing time of the message. This framework enables the monitoring and evaluation of past data, which can be valuable for enhancing the list of offensive phrases or examining user engagements over a period of time.   
  
Operational integrity is maintained by encapsulating database activities into transactions, therefore guaranteeing the integrity and consistency of the data. Error management is implemented to effectively handle any difficulties that may occur during database interactions, hence preventing data loss and ensuring the system's stability and reliability.   
  
Collectively, these elements constitute a unified and efficient system that not only detects harmful content in written form using predetermined standards, but also offers a framework for continuous adjustment and user engagement. This design facilitates the principal objective of upholding a secure and courteous atmosphere in any textual communication platform where it is implemented.

* Core Functions and Methods

**Loading Toxic Phrases (load\_toxic\_phrases Function) Purpose:** This function is crucial for initializing the application with the criteria to detect toxicity. It reads a Word document specified by the user, extracting each paragraph as a distinct toxic phrase.

Implementation: Utilizing python-docx, the function opens the provided Word document, iterates through its paragraphs, and populates a list with these phrases after converting them to lowercase for uniformity. This ensures that the toxicity check is case-insensitive and consistent.

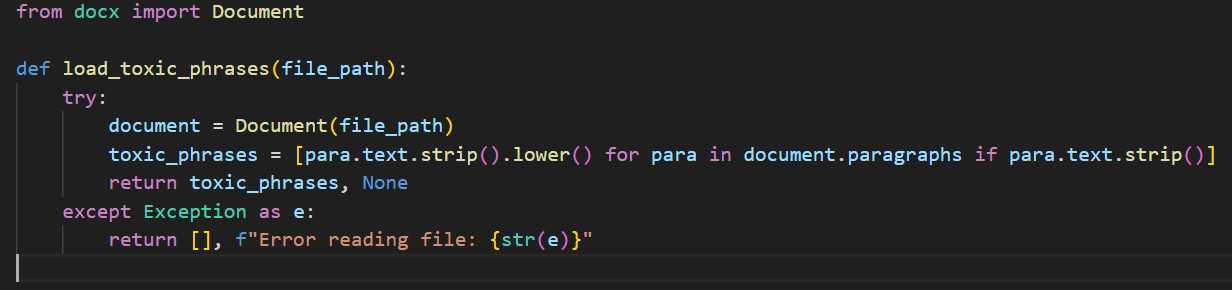


Fig. 1

**Toxicity Check in User Submissions (is\_text\_toxic Function) Purpose:** To evaluate whether any part of the user's input matches the list of loaded toxic phrases, marking the input as harmful if it does.

Implementation: The function checks if any toxic phrase is a substring of the user's input, employing a case-insensitive comparison by converting the input to lowercase before the comparison.

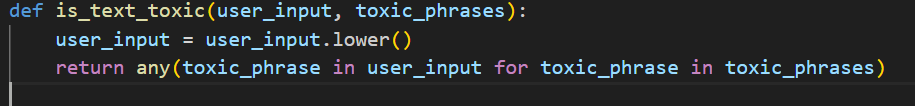


Fig.2

**HTML Form for Input and Feedback Description:** The web interface includes two main input fields—one for specifying the path of the Word document to load new toxic phrases and another for entering text to check its toxicity. The design uses Bootstrap for a clean, modern look and responsiveness.

Code Snippet: The HTML template is structured with Bootstrap components. Forms are styled for clarity, and feedback is dynamically shown using Bootstrap alerts based on the toxicity check results.

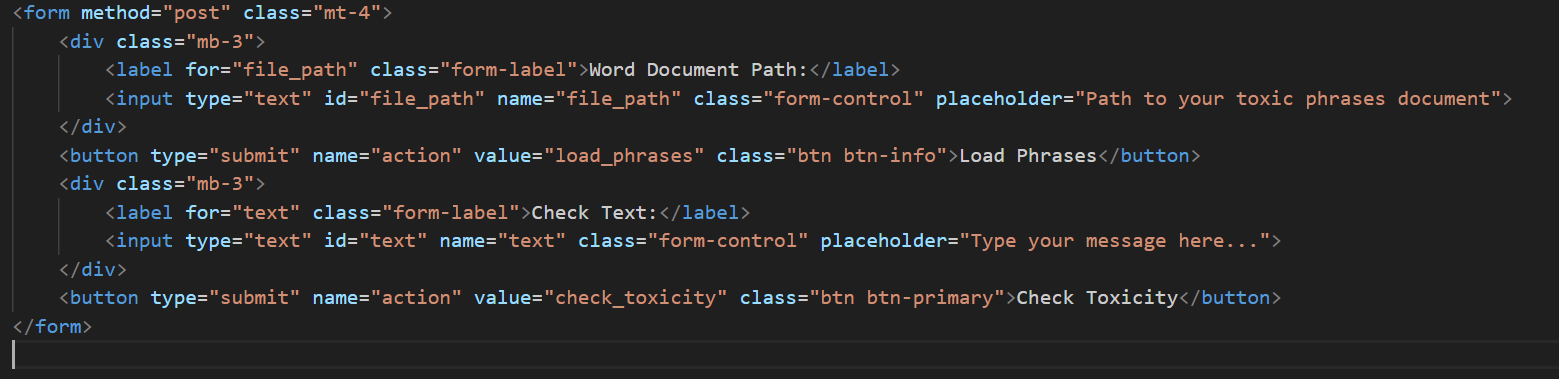


Fig. 3

**Main Route (/ Endpoint) Purpose**: To facilitate both the uploading of new toxic phrases and the checking of text toxicity based on user actions. This dual functionality is central to the application's interactive nature.

Code Implementation: The main route handles POST requests for two different actions. If the user wants to load new phrases, it updates the global list after reading the specified file. If checking toxicity, it evaluates the text and returns the appropriate result.

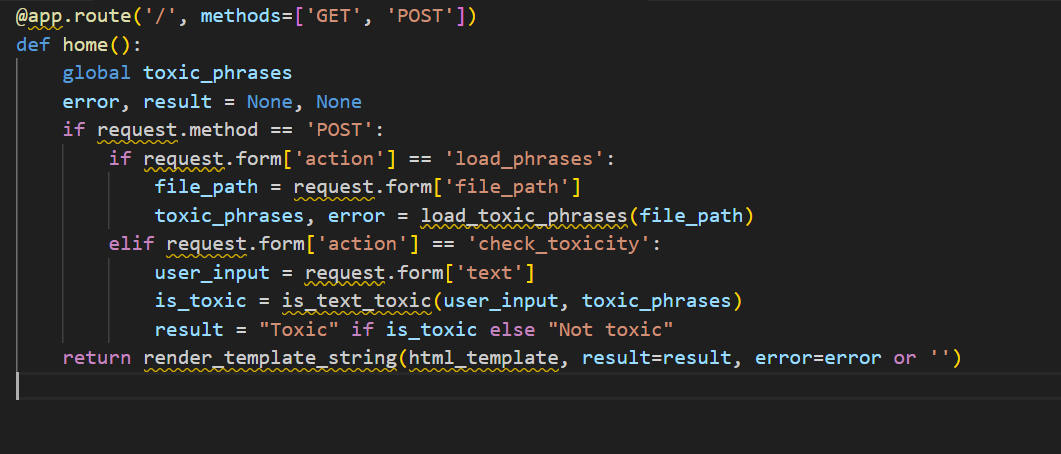


Fig. 4

File Path Validation Challenge: Ensuring the integrity and availability of the file path in contexts with highly variable routes is a crucial task. Users may input erroneous paths as a result of typographical errors, misconceptions about directory hierarchies, or disparities in file system standards among operating systems such as Windows and Unix. If not correctly managed, this might result in unsuccessful operations, user dissatisfaction, and possibly destabilize the program.   
  
Solution: In order to deal with this issue, the load\_toxic\_phrases method was enhanced with strong error handling techniques. The method is specifically built to handle errors that are associated with accessing files, such as FileNotFoundError and PermissionError. When a mistake happens, the system catches the exception and transforms it into a message that is easy for the user to understand and displays it through the interface. This method ensures the prevention of program crashes and offers users precise instructions on how to rectify incorrect input. In addition, the system records these failures for in-depth study and possible improvements to error messages, taking into account typical mistakes made by users.   
  
Issue: Case Sensitivity in Toxicity Checking The case conventions of text input and poisonous words might differ, resulting in discrepancies in the detection of toxicity. For instance, the words "Hate" and "hate" should be regarded as same. However, if not handled appropriately, they may be viewed as distinct strings, resulting in incorrect identification of harmful material, leading to false negatives.   
  
Resolution: In order to maintain uniform comparisons, both the input provided by the user and the offensive words are transformed to lowercase before any processing takes place. The normalization stage is incorporated into the toxicity evaluation process, wherein every text and phrase is converted using Python's .lower() function. The consistency guarantees that the system's sensitivity to changes in case does not impact its capacity to precisely identify harmful substance. The straightforwardness of this technique also contributes to the system's performance and dependability across various input types.   
  
User Experience Enhancements Challenge: It is essential to design an interface that is straightforward and easy to navigate for users with varying technical proficiency, particularly in applications that include sensitive activities such as toxicity assessment. Users may lack familiarity with standard error resolution procedures or fail to comprehend the significance of the feedback supplied by the system.   
  
Solution: The application utilizes Bootstrap's responsive design capabilities to generate an aesthetically pleasing and uniform layout on different devices. The alert system is utilized to promptly deliver relevant feedback on user activities, such as the successful loading of phrases or the identification of harmful content. The warnings are categorized by color: red indicates mistakes or harmful detections, while green indicates successful operations or non-toxic outcomes. All form fields have instructions and placeholders to assist users in efficiently use the system. For example, the form used to input harmful words contains a placeholder that displays a sample route, and tooltips are available for more intricate inputs. The high degree of specificity in the user interface reduces misunderstanding and assists users in resolving typical problems without requiring substantial assistance.   
  
Collectively, these answers provide a complete strategy for surmounting the difficulties encountered in the system's creation and operation, therefore improving its resilience, user-friendliness, and overall user contentment.

**Chapter 5: Future Work**

In order to optimize the functionality and increase the user experience of the toxicity detection system, it is worth considering several areas for development and extension. Here are prospective avenues for further research:   
  
Advanced techniques for Natural Language Processing (NLP)   
Identification of Toxicity within a Specific Context: Utilize advanced NLP models such as BERT or GPT to enhance the comprehension of word usage within its context. This can aid in distinguishing between stuff that is genuinely hazardous and terms that may seem poisonous when taken out of context.   
Enhance the system to provide multilingual support, allowing a wider range of users to utilize toxicity detection in their respective native languages. Using pre-trained multilingual models might be a way to improve this.   
Phrase management that is adaptable and responsive.   
Dynamic Phrase Updating: Create a user interface that allows for the addition, removal, or modification of offensive words without the need to manually edit and reload a Word document. This may involve implementing a secure administrative interface where authorized users have the ability to dynamically update the list of phrases.   
Automated Learning: Develop machine learning algorithms capable of acquiring knowledge from user input and autonomously revising the inventory of offensive expressions by detecting emerging patterns of detrimental interactions as time progresses.   
Scalability and Performance Optimization for Large Datasets: As the system expands, guarantee that it maintains good performance even while dealing with a substantially larger collection of offensive words and increased amounts of user input. Techniques like as indexing, caching, and efficient search algorithms will be of utmost importance.   
Distributed Processing: In situations when there is a significant demand, it is advisable to use a distributed processing framework or microservices architecture to conduct toxicity checks and phrase changes simultaneously.   
Improvements focused on the needs and preferences of the user.   
Customization: Enable customers to personalize their toxicity filters according to their own preferences or the rules set by their employer. Users have the ability to modify sensitivity levels or indicate certain categories of terms that they are particularly worried about.   
Feedback Loop: Establish tools for users to submit instances of incorrect identification or omission, therefore facilitating the ongoing enhancement of the system's precision and pertinence.   
Incorporation and expandability   
API Development: Create a RESTful API to enable the integration of the toxicity detection system with other programs, allowing it to be used as a service for forums, chat apps, and other digital communication platforms.   
The plugin ecosystem aims to promote the creation of plugins or extensions for widely used content management systems and communication platforms, hence promoting seamless integration and widespread usage.   
Protection and confidentiality   
Data Protection: Enhance data privacy protocols to guarantee the secure handling of user contributions and results, particularly in nations with stringent data protection regulations such as GDPR or CCPA.   
Implement logging and auditing tools to monitor usage, changes to the phrase list, and system access, ensuring compliance with regulatory standards and corporate regulations for audit and compliance purposes.

**Chapter 6: Conclusion**

Creating a toxicity detection system that utilizes predefined toxic phrases from a Word document is a notable advancement in the pursuit of establishing safer and more respectful digital communication environments. This project effectively merged a web application based on Flask with backend algorithms for analyzing user-submitted text against a list of offensive phrases, a user-friendly frontend interface for interaction, and a database for storing and managing messages.   
  
During the course of the project, numerous obstacles were faced and successfully resolved. The implementation of robust error handling effectively manages file path validation issues, ensuring the stability and user-friendliness of the application, even in the presence of incorrect inputs. To address the issue of case sensitivity in toxicity checking, a solution was implemented by converting all text to lowercase, resulting in improved accuracy and consistency of the system. Furthermore, the utilization of Bootstrap's responsive design elements and explicit feedback mechanisms greatly enhanced the user experience, effectively guiding users through their interactions with the system.   
  
In the future, there are many possibilities for improving the system. By integrating sophisticated NLP techniques, it is possible to achieve a more sophisticated comprehension of context and intention, thereby minimizing both incorrect positive and negative outcomes. Enhancing the system to accommodate multiple languages and implementing dynamic phrase management would enhance its accessibility and adaptability to diverse user requirements. Moreover, optimizing for scalability and developing an API for broader integration could extend the system's reach to a wider range of platforms and applications.   
  
Ultimately, this project has established a strong basis for identifying and handling harmful content in online interactions. This showcases the capacity to merge uncomplicated and efficient technological solutions with user-centric design to tackle crucial problems in online communication. Through ongoing development and improvement, this system has the potential to significantly contribute to the creation of healthier and more inclusive online communities.

**References**

[1] P. Dell'Aversana, "GPT-3: a new cooperation scenario between humans and machines. Benefits and limitations of GPT-3 as a coding virtual assistant," 2023. [Online]. Available: 10.13140/RG.2.2.32450.04800. February 2023]. [Accessed:]

[2] Jagran English. (2022, July 1). Google Bard vs ChatGPT: How Google is taking on OpenAI with its new software, key differences. [Online]. https://english.jagran.com/technology/Google Available: Bard-vs- Chatgpt-how-google-is-taking-on-openai-with-its-new- software-key-differences-10064291. [Accessed: February 8, 2023].

[3] A. Dasgupta, "Google Bard vs Chatgpt, and the concerns with both AI chatbots," Indian Express, 27 August 2021. [Online]. Available: https://indianexpress.com/article/explained/explained-sci- tech/Google Bard-vs-Chatgpt-and-the-concerns-with-both-ai- Chatbots-8429866/. [Accessed: 15 March 2023].

[4] J Chavez,M.R.,Butler,T.S.,Rekawek,P.,Heo,H.,&Kinzler, W.L.(2023).ChatGPT(GenerativePre-trainedTransformer): Whyweshouldembracethistechnology.Americanjournal ofobstetricsandgynecology.

[5] Liebrenz,M.,Schleifer,R.,Buadze,A.,Bhugra,D.,&Smith, A. (2023).Generating scholarlycontentwithChatGPT: ethical challenges for medical publishing. The Lancet. Digitalhealth.