June 10, 2025

C964: Computer Science Capstone

Part A: Letter of Transmittal

June 10, 2025

Stakeholders

GameStudio

25091 WGU Rd.

Dear Stakeholders,

Creating safe and respectful online environments is more critical than ever in today's digital world, especially within our game community. Currently, our in-game chat fails to catch and filter out inappropriate language, leading to complaints from concerned parents and users.

To address this issue, I propose developing a profanity-filtering chatbot. This tool automatically detects and responds to harmful or inappropriate language using a modern language model that understands context. The system logs violations securely and provides administrators with a user-friendly dashboard to monitor user behavior trends.

This solution will help promote respectful communication, foster a safer, more welcoming environment for players, and improve user satisfaction and retention. By reducing exposure to offensive language, we also protect the company's reputation and demonstrate our commitment to a positive community experience.

The project's key objective is to accurately detect and reduce offensive language in real time, supporting a healthier digital environment. We hypothesize that implementing this chatbot will significantly lower complaints related to inappropriate chat content.

The stakeholders impacted include our players, parents, game moderators, and the company at large. Players benefit from a more enjoyable experience; parents gain peace of mind; moderators have better tools to manage behavior; and the company strengthens its brand and community trust.

We will follow strict ethical and legal guidelines to ensure privacy and data security. All chat data processed by the chatbot will be handled with confidentiality, with violation logs stored securely and accessed only by authorized personnel. We will communicate transparently about how data is used, complying with all applicable regulations.

The development timeline is estimated at six to eight weeks, with costs primarily related to labor and use of existing infrastructure, minimizing financial impact.

My expertise in machine learning and data product development ensures that this project will be completed effectively and responsibly, delivering a practical solution to this pressing problem.

Thank you for your consideration. I look forward to discussing this project further.

Sincerely,

****

*****, Development Lead

Part B: Project Proposal Plan

Project Summary

This project will introduce a profanity-filtering chatbot designed to improve digital communication standards across organizational platforms. The chatbot will be built using modern NLP tools and will be deployed in a controlled environment where it can monitor user input, detect inappropriate language, and log violations for administrative review. The application will rely on a fine-tuned BERT model for classification, a PostgreSQL database for secure logging, and a graphical dashboard to display trends and summaries using Plotly visualizations.

Middle managers and IT administrators will be able to view profanity usage patterns, evaluate moderation effectiveness, and make data-informed decisions to adjust language policies or user training efforts. All components will be implemented using open-source technologies, and the system will be designed to run locally or on internal infrastructure without requiring a cloud deployment or external API dependencies.

Data Summary

The project will use a labeled dataset of text entries tagged for profanity to train the classifier. These entries will simulate user inputs typical of chat environments. The data will be cleaned and formatted to ensure accuracy, removing empty values, duplicate entries, and formatting inconsistencies. This cleaned dataset will be stored in a PostgreSQL table called labeled_tweets, which will serve as the training source for the model. During real-time operation, any user messages flagged as profane will be written to a separate violations table along with metadata such as timestamps and prediction confidence.

Implementation

Development will follow the CRISP-DM methodology, starting with data exploration and preparation. A pre-trained language model (BERT) will be fine-tuned on the cleaned dataset to classify user input based on its appropriateness. This model will then be integrated into a Python-based chatbot that accepts user input via a command-line interface (CLI). The backend will use SQLAlchemy to manage interactions between Python and PostgreSQL. SQLAlchemy will allow the application to safely and efficiently log flagged inputs into the violations table. This abstraction layer simplifies database communication and reduces the risk of SQL injection or schema mismatches. After the CLI is complete, a graphical dashboard will be developed using Plotly inside a Jupyter Notebook. This dashboard will allow administrators to visualize usage trends such as profanity frequency, model confidence levels, and input distribution. These insights will help middle management assess communication behavior over time and implement appropriate moderation policies.

Timeline

Milestone or deliverable	Project Dependencies	Resources	Start and End Date	Duration
Data cleaning	Raw open-source data	Jupyter Notebook, Pandas	07/01 – 07/05	5 days
Model training	Clean data	HuggingFace Transformers, PyTorch, pandas, NumPy, regex (re), torch	07/06 – 07/10	5 days
CLI development	Trained model	Python CLI, HuggingFace Tokenizer/Model, PyTorch, Pandas	07/11 – 07/13	3 days
PostgreSQL logging	Completed CLI	SQLAlchemy, PostgreSQL, datetime, Pandas	07/14 – 07/16	3 days
GUI development	Logging schema in place	Jupyter Notebook, Plotly, Pandas	07/17 – 07/22	6 days

Evaluation Plan

Each phase of the development process will include targeted validation steps. Data cleaning will be confirmed by checking row counts and statistical summaries before and after processing. The fine-tuned model will be evaluated using standard classification metrics: accuracy, precision, recall, and F1-score. These metrics will be calculated using a validation set withheld during model training. The command-line interface will be manually tested by submitting both valid and profane messages to ensure the classifier behaves as expected and logs violations correctly. SQL queries will be run on the PostgreSQL database to confirm that entries are accurately recorded, with correct timestamps and class labels. The final administrative dashboard will undergo user acceptance testing to ensure that non-technical users can interpret charts and retrieve meaningful insights without manual setup.

These combined evaluation methods will ensure the system functions reliably, meets business requirements, and is accessible to users with varied technical backgrounds.

Costs

 Software:

- Python 3.10.12 – Free (open source)
- PostgreSQL 15 – Free (open source)
- Jupyter Notebook – Free (open source)
- HuggingFace Transformers 4.40 – Free (open source)
- PyTorch 2.3.0 – Free (open source)
- SQLAlchemy 2.0.30 – Free (open source)
- Plotly 5.22 – Free (open source)

 Hardware:

- No additional hardware required.

 Labor:

- Data cleaning and preprocessing – 5 hours
- Model training and fine-tuning – 6 hours
- CLI chatbot development – 4 hours
- PostgreSQL setup and logging functionality – 4 hours
- GUI and dashboard development – 6 hours
- Testing and validation – 3 hours
- Total labor: 28 hours × $25/hour = $700 (estimated)

Part D: Post-implementation Report

Solution Summary

The problem addressed in this project was the detection and logging of profane or offensive language within text inputs, which is crucial for maintaining appropriate communication environments in applications such as chat platforms or customer support systems. The goal was to develop a reliable, automated profanity filter capable of identifying inappropriate language, logging violations for review, and providing insights through visualizations. The solution consisted of a (CLI) chatbot application that leveraged a fine-tuned BERT model to classify user input as profane or not. Upon detection, violations were logged into a PostgreSQL database for persistent storage. This design allowed dynamic updates and easy querying of violations over time. Additionally, plans for a graphical user interface (GUI) and interactive visualizations using Plotly were integrated to enable administrators to monitor violation trends effectively. The application successfully filtered offensive language in real-time, logged violations dynamically, and supported extensibility through modular code design, fulfilling the requirements set out in parts A and B of the project.

Data Summary

The raw data consisted of labeled tweets containing profanity, sourced from a cleaned CSV file included in the project repository. This data was imported into a PostgreSQL database and organized into structured tables to support efficient querying and logging of violations. Throughout development, the data was processed to ensure consistency and accuracy, including input validation during chatbot interactions to maintain data integrity. Logging mechanisms were implemented to capture violation details such as timestamps, user inputs, predicted classes, and confidence scores, enabling ongoing monitoring and analysis.

Machine Learning

The project used a supervised machine learning model based on a fine-tuned BERT architecture to classify user inputs into categories such as hate speech, offensive language, and neither (non-offensive). For example, the model identifies inputs like "what the hell" as offensive language with a high confidence score. The method was developed by fine-tuning a pre-trained BERT model on a labeled dataset of tweets containing examples of hate speech, offensive language, and neutral text. The dataset included 24,781 cleaned rows, with a label distribution heavily weighted towards offensive language. BERT was selected due to its ability to understand the context of language effectively, which is essential for accurately detecting profanity and offensive content.

Validation

The model falls under supervised learning because it was trained on labeled data with known categories.

Model performance was evaluated using precision, recall, and accuracy metrics on a test set of 4,957 samples. The classification report is summarized below:

Class	Precision	Recall	Accuracy
Hate speech	0.59	0.42	290 samples
Offensive	0.95	0.97	3,832 samples
Neither	0.91	0.92	835 samples
Overall Accuracy 93%			

These results show the model performs very well on identifying offensive language and neutral inputs, with somewhat lower recall for hate speech. The metrics provide confidence that the model can effectively filter and classify profane content during chatbot interactions.

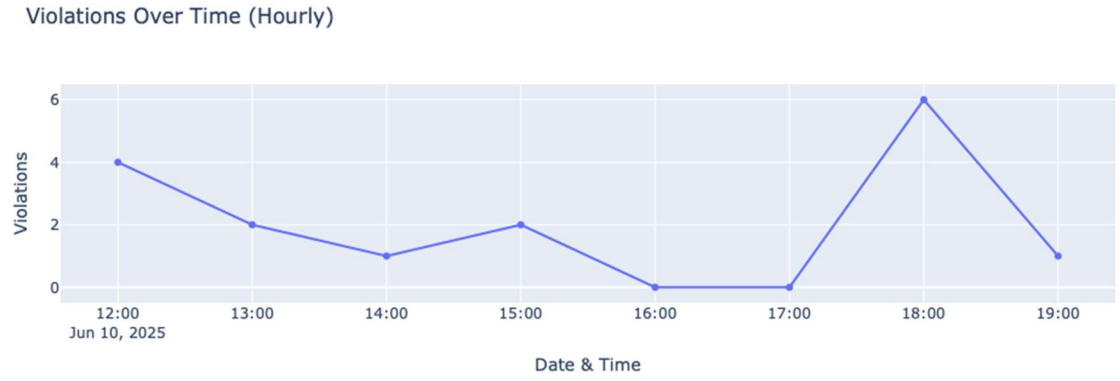
Visualizations

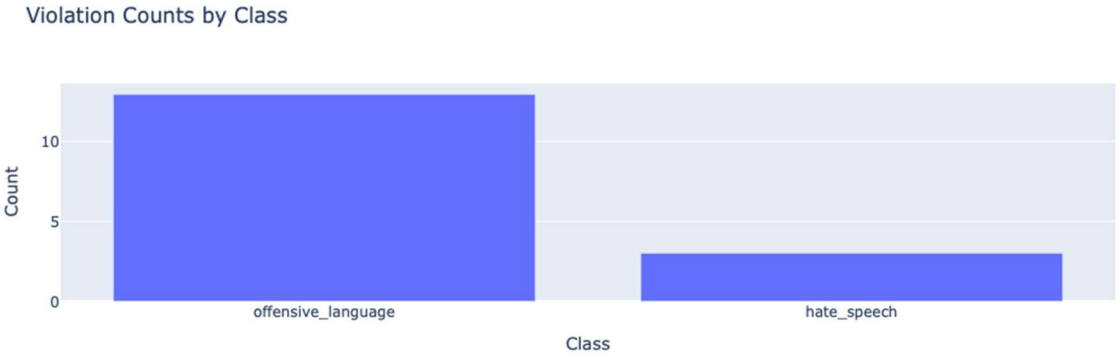
Three unique visualizations were created and integrated within the project's dashboard interface to provide comprehensive insights into profanity violations:

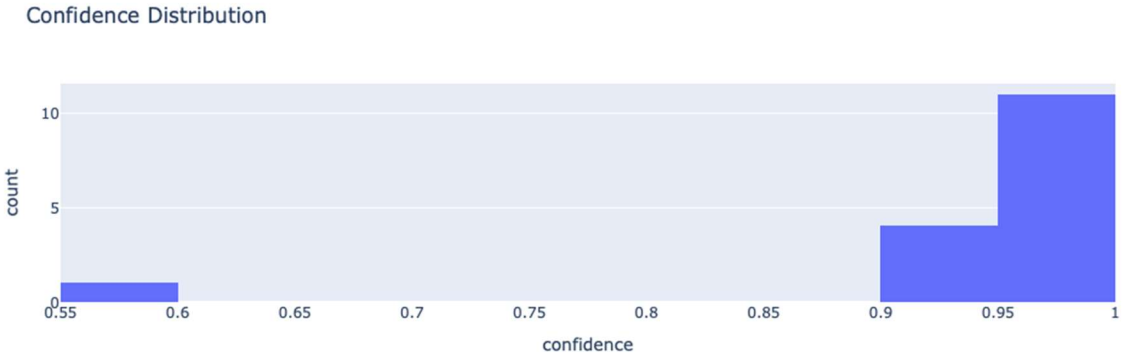
1. **Confidence Rating Histogram:** A histogram displaying the distribution of the model's confidence scores for detected violations, illustrating how confident the classifier is across different prediction instances.

2. **Violation Category Bar Chart:** A bar chart showing the count of violations broken down by category or type of profanity, enabling easy comparison of the frequency of different violation classes.
3. **Hourly Violation Trend Line Chart:** A line chart tracking the number of violations by hour of the day, revealing peak times when profane inputs are most frequent.

These visualizations collectively provide multiple perspectives on the violation data, supporting effective monitoring and analysis of the profanity detection system.







User Guide – This guide assumes you have already installed PostgreSQl and Jupyter Notebook

1. Download and unzip capstone_example.zip.
2. Go to notebook/chatbot.ipyn and run the code under RUN THIS FIRST. This will install all the dependencies for this project.
3. Open your terminal and connect to PostgreSQL as the default user by running the command: psql -U postgres
4. Create the database and user by entering the following SQL commands:

```
CREATE DATABASE capstone;
CREATE USER nick WITH PASSWORD 'gojo';
GRANT ALL PRIVILEGES ON DATABASE capstone TO nick;
\q
```

3. Connect to the new database as the user nick by running:
   psql -U nick -d capstone
4. Create the required tables by running the following SQL command:

```
CREATE TABLE violations (
   id SERIAL PRIMARY KEY,
   timestamp TIMESTAMP NOT NULL,
   user_input TEXT NOT NULL,
   confidence DOUBLE PRECISION NOT NULL,
   predicted_class_id INTEGER NOT NULL
);
```

5. Verify that the violations table is empty by running:

```
SELECT COUNT(*) FROM violations;
```

6. Open Jupyter Notebook and run the second cell in the notebook/chatbot.ipynb. Test the chatbot by entering a mix of messages, including profanities (“damn” and “hell” are included as profanities if the user does not wish to test more extreme cases) to check that violations are detected and logged correctly.
7. Open and run notebook/adminGUI.ipynb. When prompted, first enter an incorrect password such as “12” to test login failure, then enter the correct password “123.” Use the interface filters to view and analyze logged violation records.