COMMON GROUND

An AI-Driven Social Media Platform For Constructive Online Discourse

DL836 BSC [HONS] CREATIVE COMPUTING

AUTHOR: JULIA SZEWCZUK

SUPERVISOR: JOHN MONTAYNE

SECOND SUPERVISOR: CYRIL CONNOLLY

01/05/2026

Table of Contents

Acknowledgements

I would like to express my sincere gratitude to my primary supervisor, John Montayne, for his guidance, patience, and support throughout this project. His willingness to engage in collaborative brainstorming sessions and provide thoughtful, constructive feedback played a pivotal role in shaping both the scope and success of this project.

I would also like to thank my secondary supervisor and data science lecturer, Cyril Connolly, for his valuable insights, particularly regarding the communication of data findings and the effective presentation of visualisations. His passion for the field has been a genuine source of inspiration, and it is in no small part due to his influence that I have chosen to pursue a Master's in Data Analytics at the National College of Ireland.

My appreciation extends to Mohammed Cherbatji, my AI lecturer, whose teaching equipped me with a strong foundation in natural language processing and prompt engineering concepts that were directly and meaningfully applied throughout this project. His enthusiasm for emerging technologies and his commitment to continuous learning about automated systems have further reinforced my interest in data analytics and machine learning as a career path.

Finally, I would like to thank my Creative Computing peers, Aoife Lynch and Chloe Dwyer, for their encouragement and support throughout the development process.

1 Introduction and Project Context

1.1 Project Aim

The current global political trends are polarizing populations toward far-left and far-right ideologies, causing difficulties in understanding opposing perspectives. This dynamic, facilitates the growth of racist, homophobic and misogynistic discussions online by individuals who want to insult others rather than understand them. This platform aims to bridge that gap.

The aim of this project is to develop a full-stack web application that facilitates healthy online discourse through the integration of AI-driven content moderation. The platform is designed to support constructive debate by identifying potentially harmful language and encouraging users to express their viewpoints in a more respectful and inclusive manner.

The system provides users with the freedom to discuss controversial topics while offering real-time moderation feedback. When inappropriate or harmful language is detected, the system generates a safety verdict, highlights relevant moderation categories, and suggests alternative phrasings. This approach aims to guide users toward more constructive communication without restricting freedom of expression.

1.2 Project Objectives

The following objectives define the key stages required to achieve the project aim. Each objective is specific and measurable to ensure successful delivery within the project timeframe.

1. Design and produce high-fidelity wireframes for the user interface, ensuring clear navigation and usability across core user flows.
2. Develop a RESTful backend using Flask, providing fully functional API endpoints for user management, discussions, and opinions.
3. Implement a secure authentication system using JWT, including user registration, login, logout, and protected routes.
4. Design and implement a relational database schema capable of supporting threaded discussions using self-referencing relationships.
5. Develop and integrate a PostgreSQL database to reliably store and manage user-generated content and system data.
6. Integrate the OpenAI API to generate alternative, constructive phrasing suggestions for user-submitted content.
7. Fine-tune a transformer-based encoder model (XLM-RoBERTa) for multilingual, multi-label text classification focused on content moderation.
8. Evaluate the performance of the trained model using standard classification metrics, including precision, recall, and macro F1-score.
9. Integrate the trained moderation model into the backend pipeline to provide real-time analysis of user-generated content.

10. Conduct comprehensive testing, including model, endpoint, performance, feature and user testing to validate system reliability and user experience.

1.3 Success Criteria

To evaluate the success of the project, the following measurable criteria are defined. These criteria will be used in the Testing and Evaluation chapter to assess system performance against the project objectives.

User Interface & Experience

1. All wireframes are implemented into a fully functional user interface.
2. Users are able to complete core tasks (e.g. joining discussions, posting opinions) without external guidance during usability testing.
3. At least 80% of test users report positive feedback regarding navigation, clarity, and overall user experience.

Backend & System Functionality

1. All RESTful API endpoints are implemented and return valid responses.
2. Core CRUD operations (create, read, update, delete where appropriate) function correctly with appropriate permission handling.
1. User authentication (registration, login, logout) is fully functional and tested.
2. Passwords are securely hashed using industry-standard methods.
3. JWT-based authentication is correctly implemented and protects all restricted routes.
4. Token blocklisting is functional and prevents reuse of invalidated tokens.
5. Unauthorized access attempts to protected routes are consistently rejected.

Database Design

1. PostgreSQL database correctly manages relationships between users, discussions, and opinions.
2. Threaded discussions are supported using self-referencing relationships.
3. No data inconsistencies occur during CRUD operations and are validated through testing.
4. Database queries execute efficiently without noticeable delays.

AI Integration (OpenAI + Moderation Model)

1. OpenAI API integration successfully generates alternative phrasing suggestions.
2. The fine-tuned XLM-RoBERTa model is integrated into the backend moderation pipeline.
3. The system provides real-time moderation feedback during user input.
4. False positive rate remains low enough to avoid disruption to normal user interaction (evaluated during testing).

Model Performance

1. The moderation model achieves a macro F1-score of ≥ 0.80 on the test dataset.

2. Precision and recall are both ≥ 0.70 for key toxicity-related classes.
3. Performance remains consistent across multiple languages and moderation labels.
4. Harmful content is correctly identified with high reliability, as demonstrated through evaluation metrics.

Performance & Testing

1. All system components (frontend, backend, database, AI services) integrate successfully without functional failures.
2. The system passes: model, endpoint, performance, feature and user testing.

1.4 Project Scope

The scope of this project defines the boundaries of the system, clarifying what will be included and excluded to ensure the project remains feasible within the given timeframe. The project spans multiple areas of computing, including web development, user interface and user experience design, machine learning, backend systems, and database management.

The project will cover the following areas:

1. Design and implementation of the user interface, including wireframing and development of the client-side application.
2. Development of a RESTful API to support server-side functionality and system interactions.
3. Design and implementation of a relational database using PostgreSQL to manage application data.
4. Integration of external APIs, including the OpenAI API, to enhance system functionality.
5. Fine-tuning of a transformer-based model (XLM-RoBERTa) for multilingual, multi-label text moderation.
6. Implementation of a moderation pipeline to analyse user-generated content and provide real-time feedback.

The project will not cover the following areas:

1. Production-scale deployment, including cloud infrastructure and scalability considerations.
2. Real-time communication features such as WebSockets.
3. Human-in-the-loop moderation systems involving manual review processes.
4. Mobile application development (iOS/Android platforms).
5. Legal, ethical, or regulatory compliance analysis beyond basic considerations.

These exclusions are necessary to maintain a focused and achievable project scope, allowing priority to be given to core system functionality and AI integration within the available development timeframe.

2 Research and Background

2.1 Introduction

This chapter explores the evolution of content moderation, beginning with early approaches that relied heavily on human moderators and progressing toward the adoption of automated systems. It examines the factors that have driven this transition, including scalability challenges and the increasing complexity of online communication.

In addition, the chapter investigates the role of modern machine learning techniques, with a particular focus on transformer-based architectures such as Bidirectional Encoder Representations from Transformers (BERT), in the detection and classification of harmful language. These models have significantly advanced the ability to understand contextual meaning in text, making them well-suited for content moderation tasks.

The aim of this research is to identify and evaluate machine learning architectures and systems that can be effectively applied to support safe, respectful, and constructive discourse in online environments, particularly when addressing complex and sensitive topics.

Furthermore, the following research informs the development of various implementation strategies. The final methodological choice is determined by a comparative technical evaluation of the encoders, balancing domain-specific accuracy with multilingual capability.

"What technical considerations arise when implementing AI-driven content moderation for social media platforms, and how do context-specific models compare in facilitating healthy debate on controversial topics?"

2.2 Content Moderation

2.2.1 What Is Content Moderation?

Content moderation is the management of content that social media users post on digital social platforms. Media like text, images and videos are reviewed through processes like monitoring and filtering to ensure that the material shared is not promoting inappropriate, violent or hateful motives or agendas. Content moderation is applied on the basis of both legal and technical programmes and processes that aim to create a safe space for all users online. User generated content that does not meet the safety requirements but in place by enforcers can be removed from the platform and the users can face restrictions on their future actions on the designated platforms. *Seijin. (2024, December 13).*

2.2.2 The Internet Before Content Moderation

As we know, social media platforms have grown in importance and are used daily by hundreds of millions of people. These platforms have become tools for people to communicate with each other online. It was intended to facilitate easy digital communication for people, to maintain relationships with friends and family. The premise was that people could send written text messages to each other back and forth; first between each other and then with the general public in forums. *Lorenz, T. (2019, September 17).*

Due to social media growing vastly in popularity, in all areas of the world; naturally , people from different backgrounds, nationalities, cultures, views and beliefs shifted from general chat to discussions, based around sensitive topics like the current global events and their political opinions about them. *Lorenz, T. (2019, September 17).*

These platforms quickly turned from being a positive concept to a breeding ground for public discourse that facilitated cyber-harassment and created a great front for keyboard warriors to hide behind anonymous pseudonyms. *Lorenz, T. (2019, September 17).*

For example, founded in 2003, one of the oldest types of internet forums "4chan", which was an image-based bulletin board where anyone can post comments and share images anonymously which spread extremist ideologies and normalized hate and violence among users. *Lorenz, T. (2019, September 17).*

"Somewhere along the way, it began harbouring disproportionate amounts of xenophobia, racism, sexism, and nihilistic attitudes, which it eventually became notorious for. Though it's impossible to truly measure since it's completely anonymous and there are no profiles tracking user activity. It's always had trouble staying on servers due to its controversial and horrific content in general". Lorenz, T. (2019, September 17).

Lorenz T describes the content shared on 4chan to be "notorious" and "impossible to measure" due to the anonymous nature of the profiles. This indicates that there was an urgent need for solutions to tackle the hate speech problem, platform wide.

2.2.3 Evolution of Content Moderation and the Need for Automation

2.2.3.1 Introduction Of User Self-Policing (1995–2004)

Forums like 4chan relied on people who created the specific hobby forums to voluntarily step up and become "janitors" which we formally known as moderators. They didn't have real authority to stop hate speech or harassment, and in some cases, they encouraged it. In the early days, Janitors could email the founder of the website and report abuse however it was a slow process and ineffective. The platform was known for their "global rules" which mostly depended on general US laws. *Ling, J. (2023, June 5).*

"The site's Global Rules are augmented by hundreds of other board-specific rules. In practice, however, 4chan users know that the application of these rules is often arbitrary and that much depends on the moderator enforcing them. The janitors are free to deem just about anything "off topic," or to just make up their own rules as they go." *Ling, J. (2023, June 5).*

Ling J mentions that the rules are arbitrary, often overlooked and treated as suggestion rather than strict guidelines. The enforcement of the rules heavily relies on the janitor that is reviewing it. This confirms that there are global rules are not taken seriously by the janitors, there are serious content moderation inconsistencies on the platform, indicating a potential safety risk for the users.

"In the chat room, set up to help janitors discuss their moderation practices, a junior janitor asked if requesting or sharing the video of the massacre violated the rules. "It's prolly fine," replied a more senior moderator. The debate turned toward which channel was the best placed to host recordings of the livestream." *Ling, J. (2023, June 5).*

Ling J states that janitors have met to discuss the acceptability of a massacre video being shared on the platform that clearly violates the global rules. In response to that, a senior moderator has stated that "It's prolly fine,". The response from the senior moderator was short, uses slang and has a tone of uncertainty which indicates that the matter was poorly investigated, overlooked or taken seriously. It is clear that the senior moderator did not want to take full investigative measure in aid of the decision of allowing the

video to be shared on the platform. This further support that there are serious content moderation inconsistencies on the platform, indicating a potential safety risk for the users.

These voluntary janitors could either manually remove a user from the session, ban their IP address or delete the threads created. Any criticism made by the platform users towards the 4chan moderation policies was frowned upon and followed up with threats of post deletion or IP bans.

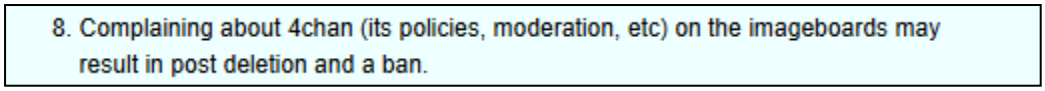


Figure 1 Rule 8 - 4Chan Moderation Board

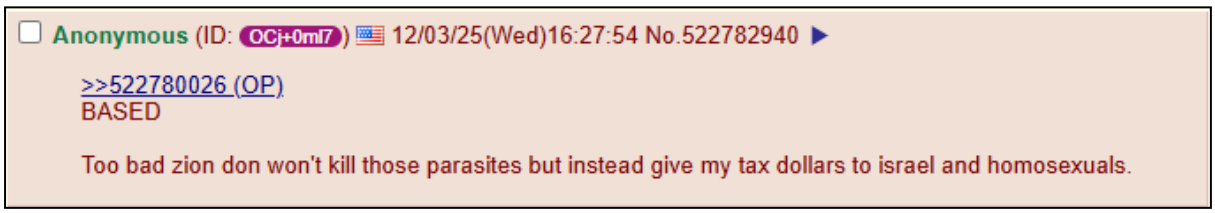


Figure 2 Example of hate speech on 4Chan

Rules - 4chan. (n.d.). https://www.4chan.org/rules

In conclusion, Lorenz T. and Ling J. believe that the moderation practices on the 4chan platform were inconsistent, poorly investigated and insufficient for handling the volume of the harmful content that was being shared on the 4chan platform.

2.2.3.2 Introduction Of Human Moderators (2004–2010)

As the internet grew in popularity, large social media platforms like Facebook and YouTube began to receive traction that was never seen before. User generated content began to flood these platforms; minimal and occasional self-policing was not enough to battle the harmful content.

Facebook began to hire graduates from San Francisco to review content, then changed their focus to outsourcing content moderators from external consulting groups.

During Facebooks early days they had content moderation rules on the basis of, no sexual exploitation, harassment or threats and graphic violence. *Sloan, S. (2021, November 11)*

"From 2004 to early 2010, Facebook not only lacked a robust team for removing problematic content but had no real content-moderation policy to speak of. Rather, platform-wide moderation was conducted by a team of a few dozen people, who were themselves guided by a one-page document of things to delete—like "Hitler and naked people"—and a general platform ethos of "if it makes you feel bad in your gut, then go ahead and take it down." Sloan, S. (2021, November 11)

Sloan S. mentions that Facebook's early moderation approaches were "guided by a one-page document of things to delete" which relates to the 4chan's "laissez-faire" global rules approach.

Human moderators hired by Facebook had to manually review posts that were flagged by users. Due to the disturbing nature of the flagged posts, moderators' mental health began to plummet.

"child pornography, murders, dismembered limbs, and other detritus of human depravity that most people never encounter". Sloan, S. (2021, November 11)

"You could not stop if you saw something traumatic. You could not stop for your mental health. You could not stop to go the bathroom. You just could not stop." Kgomo, S. (2025, February 13)

Kgomo S. clearly states that Facebook's human reliant moderation approach was de-humanising and unforgiving. The work environment did not encourage mental health, or bathroom breaks for the moderators, resulting in unsafe and unfair working conditions that ultimately led to burnout and the development of mental health conditions such as PTSD.

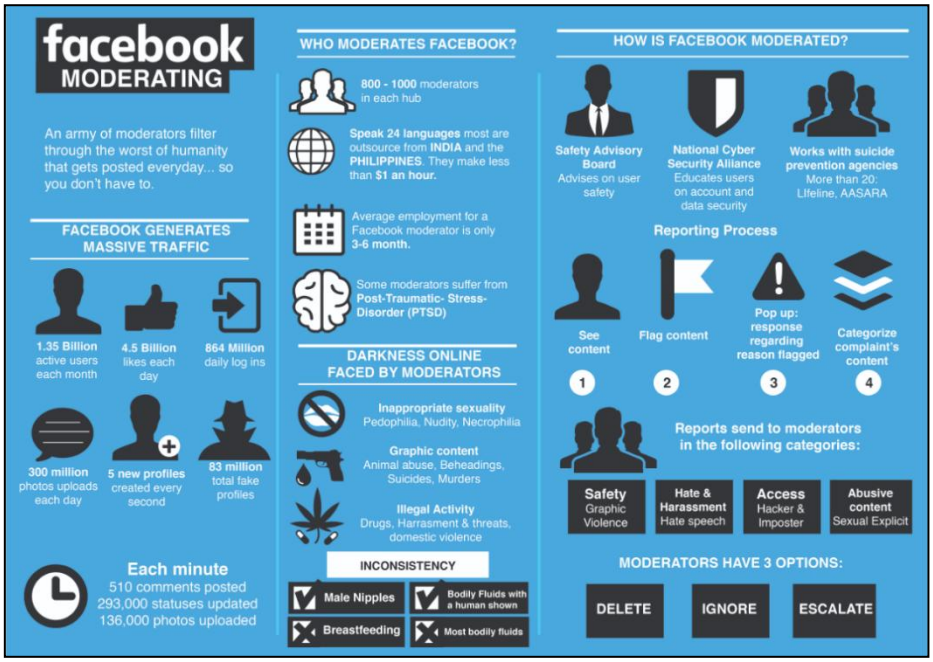


Figure 3 Infographic of Facebook's content moderation process

Michellewrites. (2015, August 21).

It is clear that according to Sloan S. and Kgomo S, an alternative approach to human moderation was needed immediately to handle the constant flow of this disturbingly harmful content.

2.2.3.3 Introduction of AI (2012–Now)

The psychological toll on human moderators the volume of user generated content made it clear that a manual approach was not scalable or sustainable. Platforms turned to artificial intelligence in the form of machine learning to automate the detection of harmful patterns at a scale no human team could manage. The following chapter explores the technical foundations of these algorithms and how they learn to differentiate between safe and harmful interactions on social media. *Sloan, S. (2021, November 11)*

2.3 Artificial Intelligence For Content Moderation

2.3.1 Machine Learning

Machine Learning is a form of artificial intelligence that focuses on creating computer models. By providing data to these models, they gain the ability to learn from information and its underlying patterns to make predictions. They can be enhanced and fine-tuned to perform specific tasks with high accuracy. There are two primary ways that models learn: supervised or unsupervised learning. *(Kufel et al., 2023)*

2.3.1.1 Supervised Learning

Supervised learning happens when a trainer provides a model with a labelled dataset, basically providing sample answers for the AI system to identify patterns in. It can function as a classifier to answer categorical questions, such as, "Is this text hate speech?" where the solution is a simple "yes" or "no."

This is ideal for quick predictions and discrete answers; the K-nearest neighbour algorithm is a classic example. *(Kufel et al., 2023)*

It can also be used for regression tasks where the model predicts a continuous value, such as a confidence score. For instance, it might determine a text is "98% likely" to be hate speech. Linear regression serves as a foundational algorithm for this approach. *(Kufel et al., 2023)*

2.3.1.2 Unsupervised Learning

Unsupervised learning is the opposite; the trainer feeds the model raw, unlabelled data that has not been pre-prepared for a specific task. Instead of predicting a specific output, the algorithm focuses on discovering inherent structures and grouping data based on shared characteristics. The main goal is to teach the machine to detect patterns and cluster information without being given a "correct" answer beforehand. The model might notice that a specific group of posts all use the same slurs, so it clusters them together as the output. *(Kufel et al., 2023)*

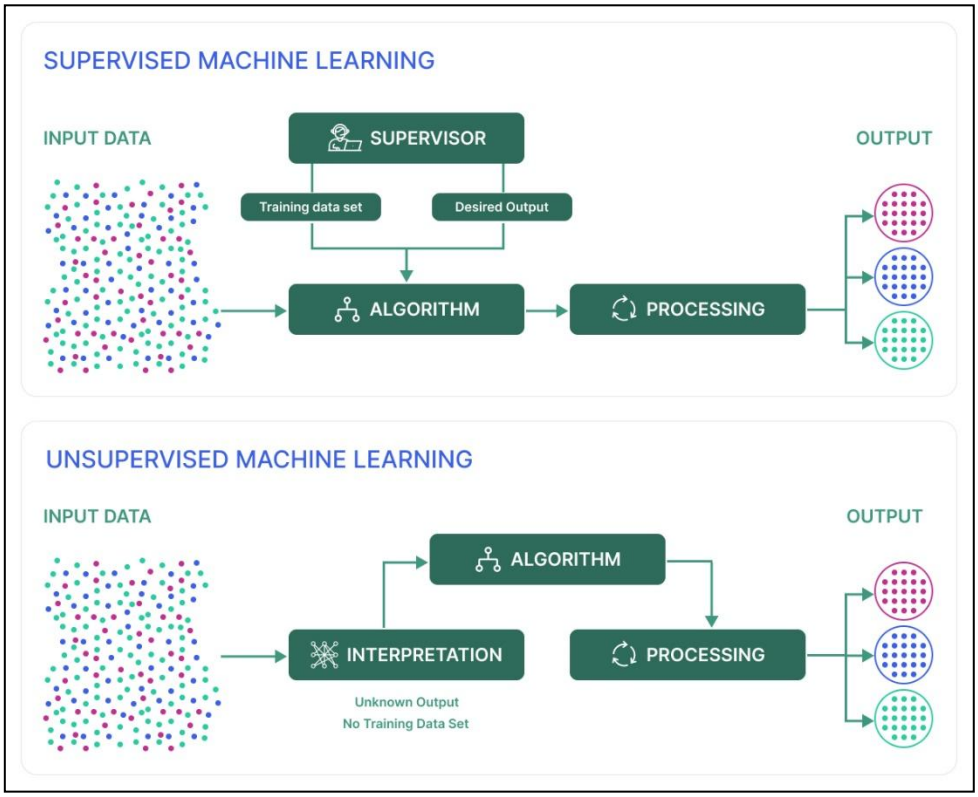


Figure 4 Diagram showcasing the differences between supervised and unsupervised machine learning architectures

(Spectrum Labs, n.d.)

2.3.2 Natural Language Processing and Deep Learning

Natural Language Processing is artificial intelligence focused on helping machines understand human language. NLP lets moderation systems actually understand what people mean by understanding context, not just words. Hate speech, threats, and discrimination are rarely straightforward, they are hidden behind sarcasm, slang and dog whistles.

While basic machine learning is powerful, deep learning takes the technology to the next level by introducing deep neural networks. Unlike traditional ML models, these networks consist of multiple hidden layers that allow the system to process data in more depth. *(Kufel et al., 2023)*

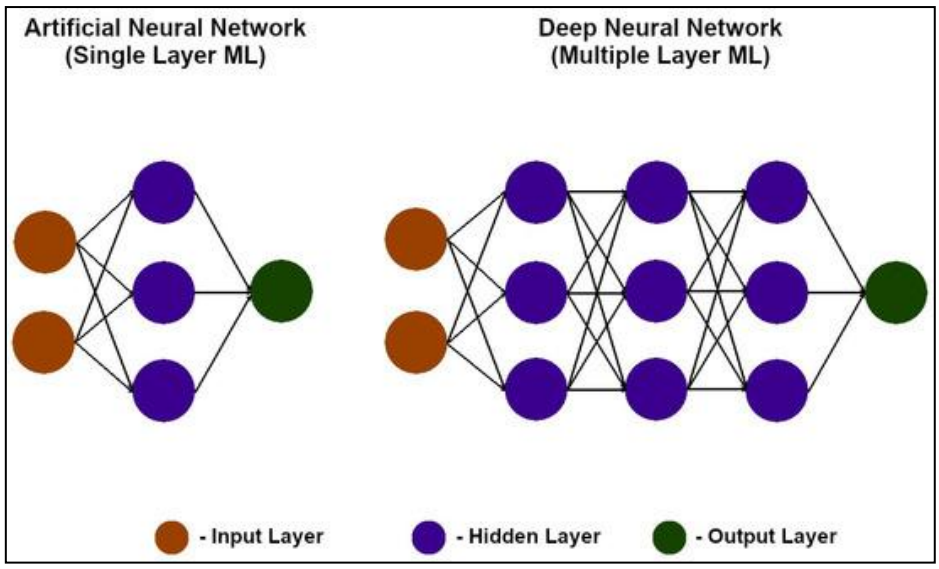


Figure 5 Neural network vs deep neural network

(Kufel et al., 2023)

This is necessary for natural language processing because human language is highly unstructured and complex. Detecting hate speech requires more than just spotting inappropriate words; it needs the model to make multiple connections to understand context, sarcasm, and intent of a user's post. Deep learning allows the system to perform these multiple passes, allowing it to grasp the subtle nuances of linguistics that basic algorithms often miss. *(Kufel et al., 2023)*

In the hidden layer stages, the model assigns weights to the different connections. The first layers might recognize individual characters or words. The middle layers notice how those words relate to each other, spotting patterns like sentence structure and the deepest layers can understand high-level concepts like aggressiveness or ill intent, even if no specific hateful words are used. *(Kufel et al., 2023)*

The main deep learning architecture for content moderation are CNN, RNN and LSTMs. Long Short-Term Memory uses sequential memory that looks at one word at a time, it also uses unidirectional context which means it reads and understands in one direction which is left to right. *(Kufel et al., 2023)*

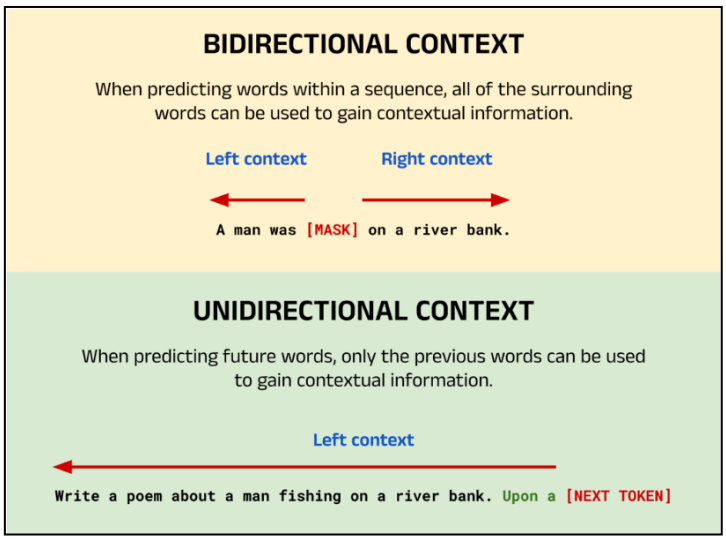


Figure 6 Infographic explaining uni and bi directional context

2.4 Technical Research

2.4.1 Encoders For Content Moderation

Context is everything in human conversations, words have double meanings and can often be used to express polar opposite emotions and intents depending on the rest of the words in the sentence, the words before and after.

"Man, I hope you go to hell!" Vs *"Hell yeah, you go girl!".*

Both of these sentences use the word “hell”, but it’s phrased differently; and have opposite meanings. An AI system needs to understand these differences to catch real threats without getting confused by similar sounding language, this is extremely difficult for a machine and is what makes moderation so hard. The system can't just search for keywords; it needs to understand linguistic patterns to learn context.

Although deep learning architectures were useful, transformers dominated the content moderation field as they are more powerful because of to their ability to process entire sequences of text simultaneously rather than word-by-word, processing words forward and backward in parallel.

By replacing the sequential memory of LSTMs with a self-attention mechanism, transformers can calculate direct mathematical connections between any two words or sentences in a post, regardless of the distance between them. Transformers reduce the high rates of false positives and increase the accuracy of predictions and hate speech classification compared to previous deep learning architectures.

Transformer architectures are constructed of an encoder and decoder, and encoder is used to classify text and make predictions while the decoder Is used to generate text. For content moderation only the encoder architecture is used.

BERT is a bidirectional encoder representation from transformers. It uses supervised learning because its’s usually fed labelled data to learn from and solve specific tasks for example content moderation related tasks. It performs exceptionally well in natural language processing tasks like sentiment analysis or text classification for hate speech in user generated social media posts because it understands text context by reading words from both left-to-right and right-to-left simultaneously. BERT uses the masked language model approach and next sentence prediction approach. It means that words in each input sequence are masked and the model is trained to predict the original values of these masked words based on the context provided by the surrounding words. This is how BERT understands context within a sentence. Next sentence prediction it understands the connection or relationship between pairs of sentences. *(Al-Hashedi et al., 2023)*

Instead of reading word-by-word in order, encoders look at the entire post at once. It understands how all the words relate to each other, not just the closest ones. They have self-attention mechanisms that let the AI look at every word in a post and figure out which words matter most for understanding the meaning of the overall post. It understands longer text without forgetting the importance of the previous words. *Horev, R. (2018, November 10)*

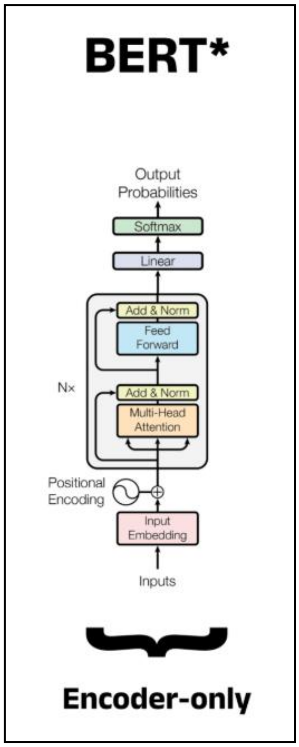


Figure 7 Infographic show casing BERT architecture

Smith, B. (2025, January 24).

Encoders don't always initially know the weights for a hateful word or phrase, so they estimate the importance of certain words in the first pass with a feed forward neural network method. When a post is first fed into the encoder system, each word is transformed into a vector embedding. This is a long string of numbers representing its coordinates in a mathematical map of human language.

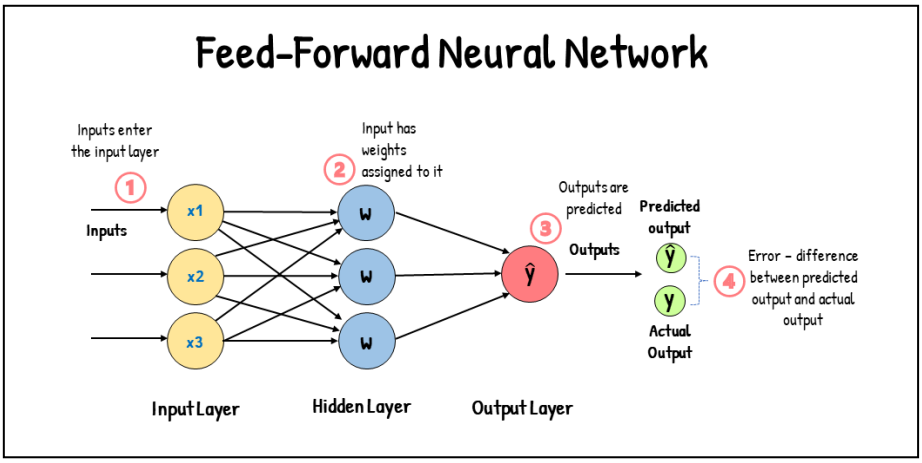


Figure 8 Infographic explaining feed-forward neural networks

Kalirane, M. (2025, May 14)

Through the multi head attention layer, the model performs a number of scaled dot-product calculations by multiplying query, key, and value matrices to see the logical closeness or semantic similarity of words regardless of their distance in a sentence of the social media post. *Kalirane, M. (2025, May 14)*

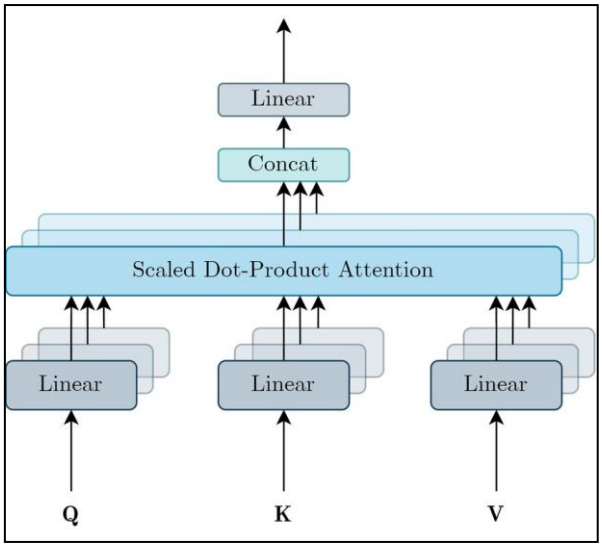


Figure 9 Infographic showasing scaled dot-product attention architecture

Or, B. (2024)

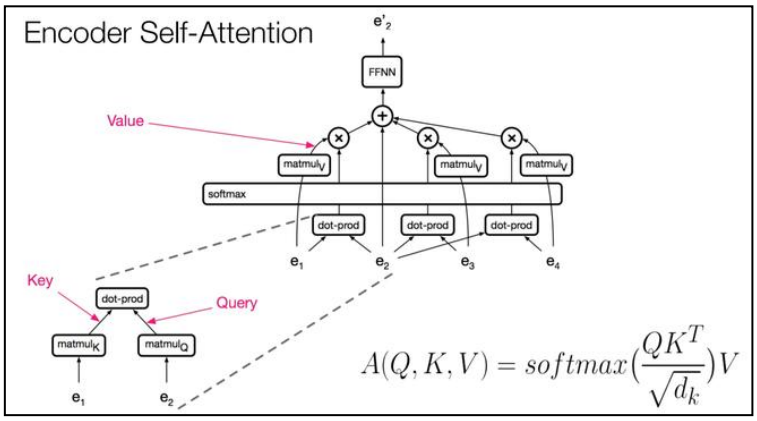


Figure 10 Infographic showing encoder self-attention architecture

GeeksforGeeks. (2025, August 23).

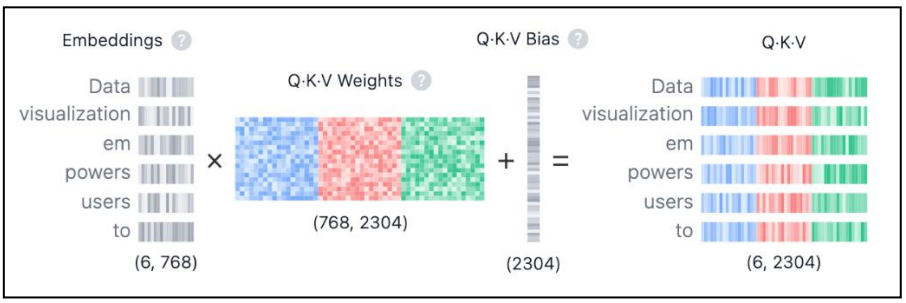


Figure 11 Infographic showing embeddings from a post being multiplied by query, key, and value weights and finally adding bias.

Transformer Explainer: LLM transformer model visually explained. (n.d.)

For example, when a long Reddit post is analysed, the self-attention mechanism allows the model to see a connection when there's a hate comment made at the very beginning of a post to a specific ethnic background mentioned at the very end, ignoring words in between. The strength of this connection is determined by these weights. It shows the importance of each embedding in a sequence relative to the other embeddings in that sequence. For example, the word "hell" might have a heavier weight than the word "the". Therefore, it won't pay much attention to It and wont associate it as part of a hate speech phrase.

During the training phase, the model learns through a process of trial and error called backpropagation. The model compares the estimate or guesses of masked words and weights it initially assigned to the correct answer and adjusts its internal weights and masked words to reduce error. If the AI fails to catch a racial slur, a trainer marks that as an error. The system then sends a signal backward through its layers and turns the weights up or down to correct the mistake. Finally, it calculates the probability of a user generated post being harmful based on the patterns of the entire message. *Kalirane, M. (2025, May 14).*

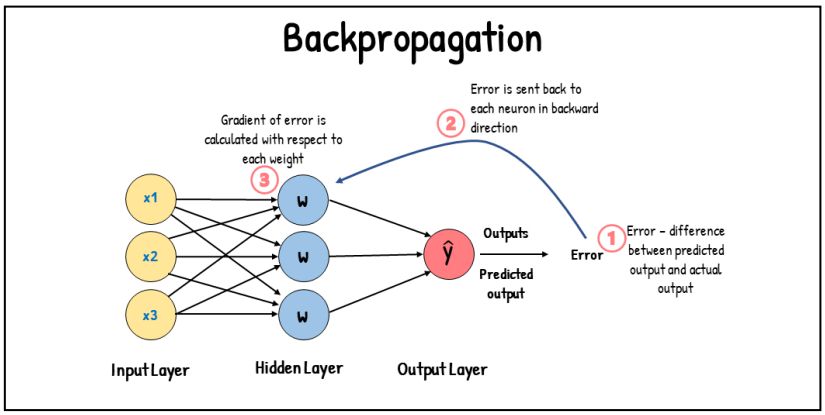


Figure 12 Infographic explaining backpropagation for feed-forward neural networks

Kalirane, M. (2025, May 14).

To get the final calculation and prediction, the encoder uses SoftMax and turns all this math into a percentage and might give a final text classification or sentiment analysis verdict based new input.

"I hope the Polish go to hell!" = 99% hate so delete or report

Vs

"Hell yeah, I love the Poles!". = 10% hate so ignore or don't report

2.4.2 General purpose vs Domain Specific Encoders

2.4.3 HateBERT

HateBERT is a domain specific encoder that was built on the foundation of BERT. It was pretrained on millions of Reddit posts that were flagged by the social media platform users from banned communities. HateBERT specialises in identifying abusive language, including offensive content and hate speech. This is because of the volume of hate speech it was already trained on like posts with; specific slang, nuances, and dog-whistles commonly found in Reddit toxic online environments. *Caselli, T., Basile, V., Mitrović, J., & Granitzer, M. (2021).*

2.4.4 XLM-RoBERTa

XLM-RoBERTa is a large general purpose multilingual masked language model. Unlike BERT, It was previously trained on 100 languages with 2.5Tb of CommonCrawl data. This model is a top choice for

content moderation systems as we know hate speech is present in all languages, therefore social media content moderation systems need to be multilingual and understand linguistic patterns for each.

XLM RoBERTa Is built on top of the BERTs foundation, therefore we already know it excels at natural language processing tasks and outperforms it. Like BERT, it doesn't have a specific speciality as it is a jack of all trades when it comes to NLP tasks. Unlike BERT, its main asset is that it has the ability to tokenize text in specific ways that the given language requires, and it understands cross-language text. It uses SentencePiece tokenizer that handles spaces, for better sentence reconstruction. Unlike HateBERT it wasn't just trained on hate speech so it doesn't automatically look for hate patterns, however it was trained on much more data and has a stronger more rigid architecture.

Transformer Encoder Comparison And Evaluation			
Title	BERT	HateBERT	XLM- RoBERTa
Core Architecture	(12 layers, 768 hidden) 110M parameters	(12 layers, 768 hidden) 110M parameters	(12 layers, 768 hidden) 270M parameters
Languages	English	English	100 languages, cross-lingual
Pre-training Tasks	Masked LM + Next Sentence Prediction	MLM only (Retrained)	Dynamic Masking No NSP
Masking Strategy	Static	Static	Dynamic
Vocabulary Size	30,000 (WordPiece)	30,000 (WordPiece)	250,000 (SentencePiece)
Training Data Size	16 GB (BooksCorpus + Wikipedia)	71 million tokens (RAL-E dataset)	2.5 TB (CommonCrawl - CC100)
Data Source	Formal (Books, Wiki)	Social Media (Reddit banned communities)	Diverse, massive-scale web crawl
Best Use Case	General English NLP tasks	Abusive language/Hate speech detection	Multilingual or cross-lingual apps

Figure 12 Evaluation table comparing BERT, HateBERT and XLM-RoBERTa

Comparison of BERT, XLM-RoBERTa, and HateBERT specifications. (2025, December 22). Hugging Face Model Hub. Table created by agentic AI assistant with Kimi-K2-Instruct-0905 via Groq. https://huggingface.co/

(BERT 101 - State of the Art NLP Model Explained, n.d.)

2.4.5 Conclusion

While we think that HateBERT would be the ultimate model for content moderation and hate speech on a social media board or discussion forum, it does not classify hate speech well in languages other than English. We would then assume that XLM-RoBERTa would be superior for this kind of platform because it understands the nuances of 100 languages; however, it performs better at understanding the nuances of normal, everyday language rather than hate speech in particular.

2.4.6 Implementation Strategies

2.4.6.1 Strategy 1

Is using the model fusion method that combines the architectures of both HateBERT and XLM-RoBERTa models, merging them to create one new model. This known as stacked generalization.

Brena, R. F., Aguileta, A. A., Trejo, L. A., Molino-Minero-Re, E., & Mayora, O. (2020). Choosing the best sensor fusion method: a Machine-Learning approach. Sensors, 20(8), 2350. https://doi.org/10.3390/s20082350

2.4.6.2 Strategy 2

Is using the ensemble method that involves taking the median score from the SoftMax layers of both the HateBERT and XLM-RoBERTa models and calculating the average prediction score.

Naderalvojoud, B., & Hernandez-Boussard, T. (2024, January 11). Improving machine learning with ensemble learning on observational healthcare data. https://pmc.ncbi.nlm.nih.gov/articles/PMC10785929/

2.4.6.3 Strategy 3

Is to fine-tune XLM-RoBERTa on hate speech from a handful of different languages. However, this can be highly resource-intensive, consuming significant time for training, as well as computing and GPU power. It would also require extensive data collection of hate speech across different languages, if no such data set is available, along with preparation and preprocessing for labelling and tokenization.

2.4.7 Verdict and Strategy Choice

In conclusion I will be implementing strategy 3 previously stated as the fine-tuning XLM-RoBERTa approach. Although it requires greater computational resources and data preparation, it is the most likely to perform well across hate-speech in multiple languages. Despite its higher cost, I think that option 3 represents the most methodologically sound choice for this task without adding unnecessary architectural complexity.

The comparison of models revealed important trade-offs between domain specificity and multilingual capability. These insights guided me to select XLM-RoBERTa model and fine-tune it for this project.

2.5 Challenges in Content Moderation

2.5.1 Ambiguous Language

The biggest challenge is that strong opinions, debates about politics, social justice, and human rights, often use intense language without actually being hateful or harassing. Forum users in passionate conversations might use exaggeration, sarcasm, or confrontational tone in their discussions but that doesn't automatically mean they intent harm.

Simple AI systems that just look for certain words or phrases often get this wrong. They might flag someone's legitimate political criticism as "harassment" when it's actually just passionate debate. This kind of mistake risks silencing people who have every right to speak their minds. This is the reason why many social media users, especially American disapprove of content moderation as they feel they have a right to free speech, regardless of if it's offensive or harmful. *(Al-Hashedi et al., 2023)*

2.5.2 Unlabelled Data

Not all training data is pre-classified and ready for the model to be trained on, often times it's the data analysers or model supervisor responsibility to do so. However, if they let their personal opinions slip into their professional work while labelling the training data, they might make the model biased, and potentially more or less sensitive to moderating hate speech.

When it comes to using flagged data scraped off the internet, depending on what society considers acceptable at any given moment, what culture the users are from, what their sarcasm threshold is and how individual people interpret words, people might not always agree. If you show a crowd of forum or social media users a borderline comment and ask, "Is this harmful?", they might have split opinions. *(Al-Hashedi et al., 2023)*

That is when these disagreements get resolved by majority vote, the unpopular opinions get ignored. AI systems trained this way might accidentally build in biases. This is especially hard when dealing with controversial topics as some users are more sensitive to certain topics and are more likely to flag certain terms compared to other users. *(Al-Hashedi et al., 2023)*

2.5.3 The Imbalance Problem

Although harmful content is present on social media it's less frequent than neutral content. Most posts are neutral, positive or ambiguous, so training datasets end up being mostly harmless examples. AI

systems trained on this imbalanced data can look very accurate overall but actually have weaker performance when identifying threats or abuse, which are the rare harmful things they're actually supposed to catch.

On the other hand, content moderation system to put more emphasis on strict screening we can accidentally over predict and flag a lot of legitimate posts. It's hard to get the balance right. *(Al-Hashedi et al., 2023)*

2.5.4 Language Keeps Changing

As social movements and new trends arrive, users create new slang and coded language to get around moderation filters. Often times words that previously meant something harmless sometimes pick up harmful meanings. This is an issue because AI systems that don't get updated regularly and find it difficult to recognise the changes fast enough to keep up with linguistic social media trends. *(Al-Hashedi et al., 2023)*

2.6 Conclusion

Hate speech is a true social media problem that has been affecting social media users online since the beginning of the internet. The need and demand for moderating user generated content was prominent since the start of the internet, yet its implementation wasn't easy or smooth sailing. The shift from human to machine automation is needed due to the sheer nature and volume of user content generated on social platform at every second.

I believe the best move forward for moderation is to utilise modern technology like Encoder based transformers due to their superior for NLP text classification tasks for content moderation, specifically in hate speech detection. Although they are not perfect, they are a significant improvement over CNN, RNN, and LSTM architectures for rapid online discourse. Their bidirectional processing, tokenization, embedding, and self-masking architecture to understand social media posts like humans is the future of facilitating a safe space for people to have healthy discussions online.

Building on this research, the next chapter translates these theoretical insights into practical requirements, focusing on user needs, system functionality, and design considerations.

3 Requirements Analysis

3.1 Introduction

Requirements analysis is an important stage in the software development lifecycle, as it ensures that the final application aligns with user needs and project objectives.

This chapter begins by examining similar platforms to identify common features, strengths, and limitations. These insights aid with the creation of functional and non-functional requirements for the platform, supported by personas and a use-case diagram. The goal of this chapter is to establish a clear and structured specification that will guide the design and implementation of my system.

3.2 Requirements Gathering

This chapter includes highlighting strengths, weaknesses, moderation strategies and overall user experience of competitor applications. The analysis includes the observation of navigating, viewing and creating discussions, threads, nested replies and discovering how user generated content is moderated.

3.2.1 Competitor Analysis

3.2.2 4Chan

As previously mentioned in the research chapter, 4chan is an anonymous image-based bulletin board platform established in 2003. It remains active, attracting a large user base of 20 million monthly visitors. The platform facilitates discussions through topic-specific “boards” created by administrators. Within these boards, users create threads that support short-lived, anonymous discussions.

3.2.2.1 User Experience and Interface Design

The platform adopts a minimalistic design based primarily on plain HTML with limited CSS styling. While this approach preserves a nostalgic aesthetic, it results in a cluttered and unintuitive interface. Threads are difficult to follow due to the lack of clear visual hierarchy, inconsistent formatting of posts and images. The anonymous identifiers and unannotated hyperlinks feel unsafe and unintuitive. Navigation is further hindered by dense layouts, cluttered cards and poorly structured form inputs, which can reduce usability, particularly for new users.

Why do Reddit and 4chan have such bad user interfaces? (n.d.). Quora. https://www.quora.com/Why-do-Reddit-and-4chan-have-such-bad-user-interfaces

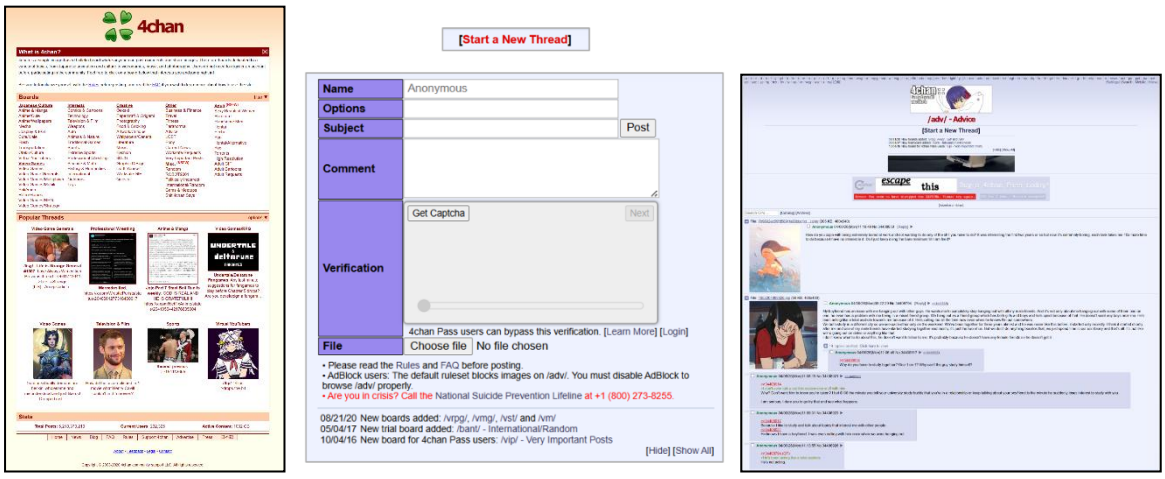


Figure 13 Screenshot of 4chan home page

Figure 13 Screenshot of 4chan create new thread form

Figure 14 Screenshot of 4chan threaded conversation inside advice board

3.2.2.2 Moderation And Rules

4chan does not used AI-powered moderation systems and the site relies on manual review and removal of posts driven by janitors. 4chan establishes a list of global rules that apply to users, boards and threads. In addition, specific boards have additional rules that apply and disciplinary bans occur when rules are broken. Rules are long lists that are confusing to read and include special characters and command like architecture that might be unfamiliar to some individuals.

The manual moderation approach is labour intensive and unreliable, potentially causing false negative examples of hate to be missed. This ultimately makes the moderation approach unreliable and potentially unsafe. Due to the anonymous nature of the platform users with poor or hateful behaviour are not distinguishable and cannot be avoided/monitored or penalised by other users other than the janitors themselves.

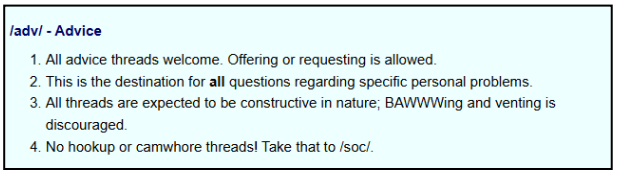


Figure 15 Screenshot of 4chan global rules

Figure 16 Screenshot of 4chan global bans

Figure 17 Screenshot of 4chan advice board specific rules

3.2.2.3 Findings And Solutions

After analysing 4chan, three primary weaknesses were observed: absolute anonymity, inadequate moderation, and a disorganized user interface. I believe that requiring non-anonymous profiles fosters a sense of responsibility and accountability, incentivizing users to maintain a positive persona and reputation on the platform.

To address these shortcomings, my platform implements three core solutions:

1. Automated content moderation to prevent the oversight failures found on 4chan.
2. Creating an intuitive interface to aid in simple navigation.
3. Requiring registered accounts allows for an upvote/downvote mechanic. This encourages users to strive for an "elite" status while minimizing harmful content through community-driven accountability.

3.2.3 Reddit

Reddit is also a discussion-based social platform just like 4chan, established in 2005. It remains active, attracting an enormous user base of approximately 1.36 billion million monthly visitors. The platform is built around users account and features topic specific communities known as subreddits, and a voting system that helps surface popular content.

3.2.3.1 User Experience and Interface Design

Reddit has a much more structured and accessible interface than 4chan, with clearly separated communities, threaded discussions that help users navigate content more easily. The platform is built around accounts rather than anonymous users, which makes it easier to follow user activity, build reputation, and participate in topic specific spaces called subreddits.

Reddit has evolved beyond basic HTML and static CSS, adopting a token-driven design system that prioritizes structure and clarity. By utilizing a typography hierarchy, modern Flexbox layouts, and component-based styling, Reddit creates a consistent visual language specifically through defined card elements that offer more predictability than 4chan's free-form boards. Integrated transitions and animations provide a fluid, modern feel, while the use of intuitive icons and buttons ensures that navigation is intentional and accessible even for first-time users.

Jeon, S. (n.d.). Streamlining Reddit UI components: The Reddit design system. Medium. https://medium.com/@sehyunjeon/streamlining-reddit-ui-components-the-reddit-design-system-a90189fd9c38

This architecture results in a highly organized user experience, optimized for browsing and long-form engagement. In contrast to 4chan's fast-moving and often cluttered environment, Reddit's design feels stable and readable, effectively shifting the focus toward sustained community discourse.

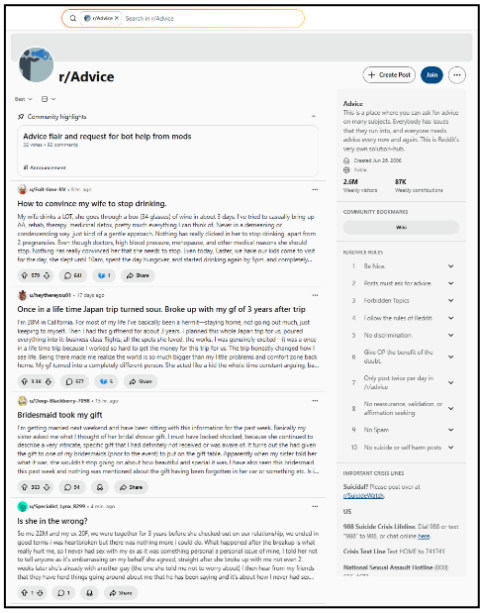


Figure 18 Screenshot of Reddit advice subreddit

Figure 19 Screenshot of 4chan threaded conversation inside subreddit

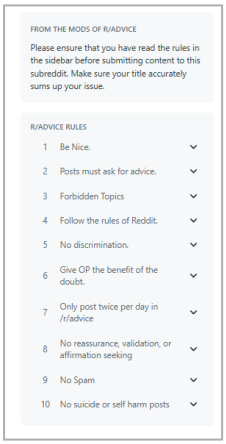
3.2.3.2 Moderation And Rules

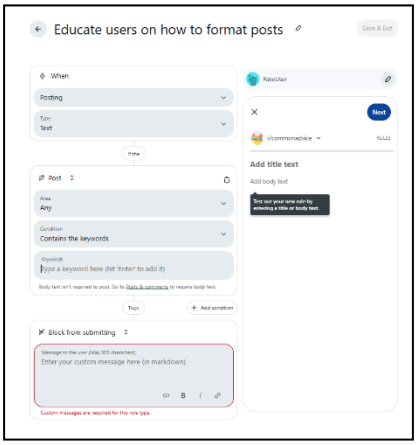
Reddit uses a hybrid system of automated and human-in-the-loop moderation tools for both global and subreddit levels. These include for example: A large language model powered tool that automatically flags content likely to contain harassment and a highly bot used by volunteer mods to automatically remove based on specific triggers like keywords.

Beyond global rules, mods create and enforce their own community guidelines and moderation on Reddit is highly dependent on individual subreddit rules and volunteer moderators, which can lead to inconsistency in enforcement across the platform.

Digilogy. (2025, April 7). Reddit introduces AI-powered community moderation tools. https://news.digilogy.co/reddit-introduces-ai-powered-community-moderation-tools/

While this community-led model improves accountability compared with anonymous boards, it can still struggle with harmful content and ideological echo chambers. These observations make Reddit a useful competitor to analyse because it demonstrates both the advantages of identity-based participation and the limitations of decentralized moderation.





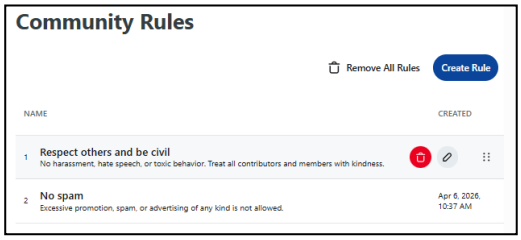


Figure 20 Screenshot of Reddit advice subreddit rules

Figure 21 Screenshot of Reddit create conditional rules form

Figure 22 Screenshot of Reddit create community rules edit form

3.2.3.3 Findings And Solutions

After analysing Reddit, three primary weaknesses were observed: the excessive administrative burden of content review, the high barrier to entry for setting up effective moderation, and the overwhelming complexity of configuring fragmented community rules.

To address these challenges, my platform implements four core solutions:

1. Minimal admin interference, just accepting and rejecting users of membership no review of flagged content or active content moderation, significantly reducing responsibility and task overwhelm.
2. Global rules, that apply to both discussion and opinions, eliminating the need for fragmented, community-specific rulemaking.
3. Fast, easy and minimal effort process of creating discussions with no technical setup.
4. Flagged content is highlighted to the creator of the text but not forcibly removed. The user is made aware of the repercussions of posting harmful content and a civility score is kept for accountability's sake.

3.2.3.4 Conclusion

By conducting the competitor analysis of both platforms and their structural extremes, I have identified a "common ground" that leverages the speed of a bulletin board with the accountability of a modern social media platform.

My platform eliminates the chaotic interface, total anonymity, and labour-intensive manual moderation found on 4chan. Simultaneously, it solves Reddit's core frustrations around the time-consuming, rule-based setup and the administrative exhaustion caused by constant manual oversight. The result is a user experience that prioritizes both user responsibility and ease of use.

Conclusion Of Features To Implement In Project

1. Accountability With Civility Score

   By requiring registered profiles, I replace 4chan's anonymity with a Civility Score system. This is a score calculated from how safe the submitted text is based on if and what harmful categories were triggered. This fosters a drive for "legend" reputation, but rather than deleting flagged content, it highlights it to the creator and those around them. This shifts the burden of behaviour from a moderator's "delete" button to the user's personal accountability.

2. House Rules

   To avoid the tedious setup of Reddit's subreddits , my platform uses House Rules. This ensures that all discussions follow the same standards for opinions and discourse, removing the need for users to learn or admins to implement discussion-specific rules.

3. Passive AI-Moderation

   I address the administrative burden of both the competitors by implementing AI moderation to track toxicity and provide real-time feedback. Human admins are freed from the task overwhelm of reviewing every flag, focusing instead on high-level membership management (accepting/rejecting users) while the AI maintains the site's baseline safety.

4. Optimized UX for Discourse

My platform adopts the user centric design methods of modern Reddit such as intuitive navigation and clear visual hierarchy but maintains the low maintenance of 4chan. Users can launch and engage in discussions instantly with little to no technical setup.

These insights will be used to develop a platform that feels as easy to set up and spontaneous as 4chan but as structured and readable as Reddit.

3.3 Requirements Modelling

This chapter includes functional and non-functional requirements, personas and use-case diagrams used for this chapter. High-resolution versions of the diagrams are found on my Miro board. https://miro.com/app/board/uXjVJoNoh0o=/?share_link_id=495971344352

3.3.1 Personas

Personas are fictional characters created to help designers and developers understand and empathize with the target audience that they think would use this type of application. These characters have an overview of their personality, detailed demographics, their own unique pain-points, frustrations, needs, goals and personal motivations to use this type of platform.

I have created three fictional personas, these people vary in age ranges, family life, political stand points, career paths and stages of their lives.

Figure 23 Conservative persona diagram

Figure 24 Progressive persona diagram

Figure 24 Pro self-governance persona diagram

Across the three personas, a consistent pattern is clear. They want a discourse platform that enables respectful, structured interaction across ideological differences without the hostility, distortion, or echo chambers typical of mainstream social media. Each persona values the ability to express their views without being dismissed or attacked, while also gaining insight into how others think whether it be across political, cultural, or lifestyle differences. Additionally, there is a shared desire for tools that help filter between overall positive and negative discussions while still exposing them to users with diverse viewpoints.

3.3.2 Functional Requirements

Functional requirements when referring to application development take in the persona's requirements, needs, motivations and the collection of features gathered from the competitor analysis . They are then combined to create desired features for the platform. Development can be time consuming and prioritising certain features is necessary. Therefore, the priorities are defined by either high, medium or low priority.

No.	Requirement	Description	Priority
1	User Authentication	Individuals can register for an account. Users can log-in to their existing account.	High
2	Discussions	Threaded conversations that can be either private or public.	High
3	Roles	Users can either be a Regular or Admin user inside a discussion.	High
4	Membership & Privileges	In private discussions, users request to join. The creator of the discussion automatically becomes admin. Admins have the privilege to accept, reject or block a user from their discussion.	High
5	Opinions	Users can share their opinions within a discussion they're a member of.	High
6	Nested Opinions	Users can reply to other users' opinion.	High
7	AI Harmful Language Detection System	Fine-tuned XLM-RoBERTa system flags harmful language. Alerts users of the categories flagged and if the text is safe to submit.	High
8	AI Alternative Response Generation System	OpenAI powered system suggests alternative phrasings for flagged content by XLM-RoBERTa.	High
9	Civility Score	This is a score calculated from how safe the submitted text is based on if and what harmful categories were triggered.	High
10	Voting	Users can see their own vote results. Users can upvote or downvote other users' profiles.	High
11	House Rules	Global rules and terms and conditions displayed to encourage respectful discourse.	High
12	Search & Filtering	Users can search for discussion by title and description. Users can filter discussions by open, closed, private and public.	Medium
13	Discussion Life Cycle	Discussions are set to be open for a certain period of time.	Medium
14	Perspective Tags	Labels indicating the political or ideological perspective of a user.	Low
15	Viewpoint Tags	Users select tags that they feel represent their views when creating discussions and sharing opinions. These include for, against, centre and undecided.	Low

16	Analytics	Discussions receive reports on discussion quality, tone, and sentiment metrics.	Low

3.3.3 Non-Functional Requirements

Non-functional requirements are functions and features that aren't physical components but rather characteristics that improve the performance of the application. This usually indicates if an application is up to standard when it comes to reliability, usability, scalability and other characteristics of an industry standard software.

No.	Requirement	Description	Priority
1	Responsive	The platform needs to be compatible with all screen sizes such as monitors, tablets, phones etc.	High
2	Compatible	The application should work in all web browsers eg: chrome, edge, safari	High
3	Consistency	Optimized UX for discourse should maintain consistency in fonts, sizes, colours, and layouts across all pages.	High
4	Speed	Response time of any request should be less than 2 seconds.	Medium
5	Scalability	Should be able to handle a large dataset with a volume of users and discussions and other entities.	Low

3.3.4 Use Case Diagram

The use case diagram introduces a persona to make mock interactions with the platform. This is done to visualise how a potential user would interact with the platform, with goals set in mind and monitor what steps they take to reach to achieve their set goals.

An analysis of each event is documented and compiled into a list of features that need to be implemented somewhere in the development stage for these tasks to be completed as quickly, efficiently and as easily navigable as possible.

Implementing these features, keeping in mind that the user experience and user interfaces need optimum user interactivity and navigation assets like layouts, menus, buttons and their placements, which are crucial for smooth operations.

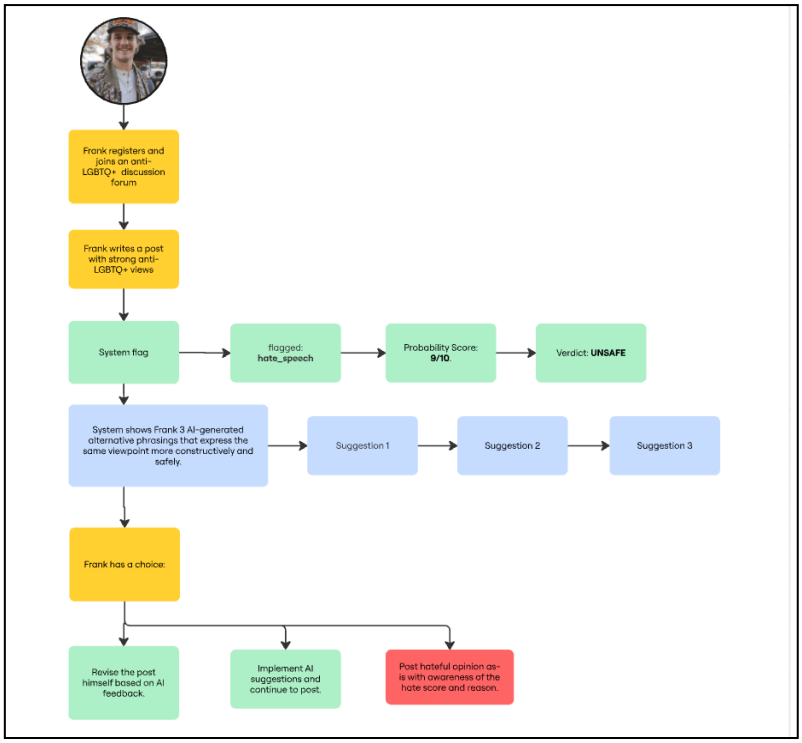


Figure 25 Persona Use Case Diagram - Submitting An Opinion Inside A Discussion

Frank registers and joins an anti-LGBTQ+ discussion

Frank writes a post with strong anti-LGBTQ+ views

Before submission, the system analyses the post

System flags: **Toxicity Probability Score: 9/10**

Frank gets immediate feedback showing:

- His post is considered toxic.
- Toxicity Probability Score: **9/10**.
- Reason why it was flagged because it contained **"Violence".**
- The verdict being **"Unsafe"**
- System shows Frank 3 AI-generated alternative phrasings that express the same viewpoint more constructively and safely.

Frank can:

- Revise the post himself based on AI feedback.
- Implement AI suggestions and continue to post.
- Post hateful opinion as-is with awareness of the hate score and reason.

3.4 Conclusion

This chapter translated the personas desires and insights gained from competitor analysis into a structured set of system requirements. Functional and non-functional requirements were defined to guide development, while personas and use-case diagrams provided a user-centred perspective on system interactions.

These requirements form the foundation for the design phase, ensuring that architectural and interface decisions are aligned with user needs and project objectives. The following chapter father builds specification by outlining the system's architecture, interface design, and overall structure.

4 Design

4.1.1 User Interface Design

The High-resolution versions of the diagrams such as user flow diagrams, final wireframes/mock-ups with rationale for UX/UI decisions, storyboards, style guide can be found on Figma.
https://www.figma.com/design/jM6Tnhsny1KJfTBluy91Cn/Thesis?node-id=0-1&t=a43fgLrj9Y2RGS2y-1

4.1.1.1 Initial User Flow Diagram

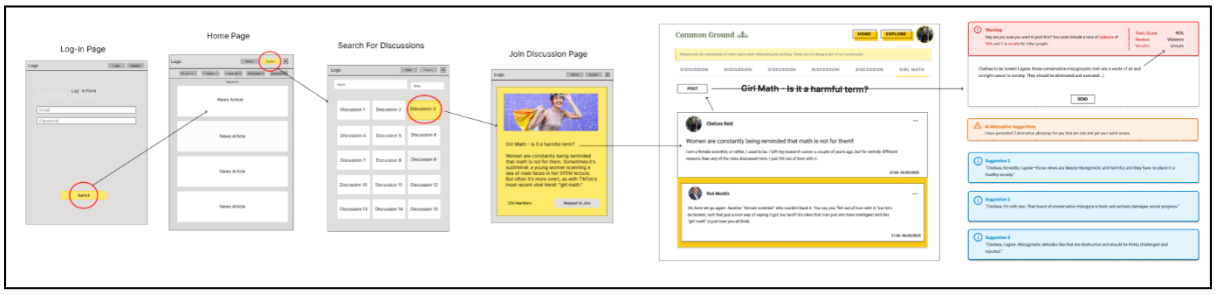


Figure 26 Screenshot of user flow diagram - Submitting An Opinion Inside A Discussion

This user flow diagram incorporates low and high-fidelity wireframes to display the process of a user logging-in, navigating from the find all discussion page to searching for a specific discussion they would like to participate in. I decided to shift the focus of the flow more on the user interacting with the moderation system and how it responds to user input. The main focus was the user receiving a negative score and receiving AI generated suggestions to better their argument in a non-harmful way. Further on I stylistically steered away from the yellow tones and decided on a more monochromatic colour palette.

4.1.1.2 First Iteration

Initially the application had more focus on being very politically motivated and the theme continued in my choices of contents I designed for the user. This design had a home screen filled with sophisticated articles on topics like economics, education, child development etc. I was advised by my supervisor that the atmosphere and user experience of the application felt bland and uninspiring and that pulling real articles made the platform feel like any other.

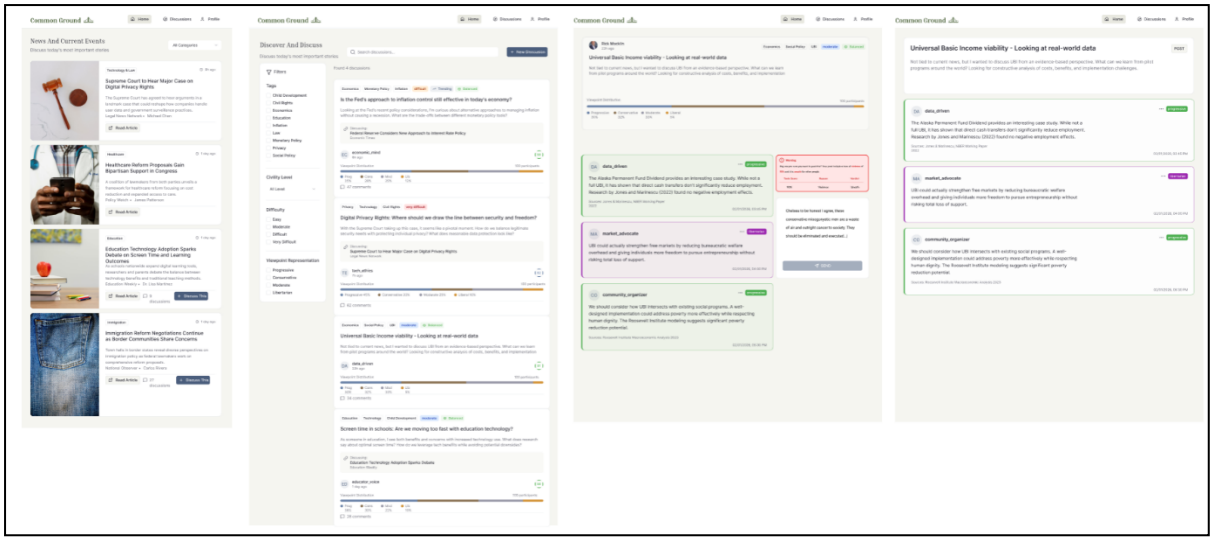


Figure 27 Screenshot of iteration 1 user interface designs, home page, search page, discussion show page and discussion detail page

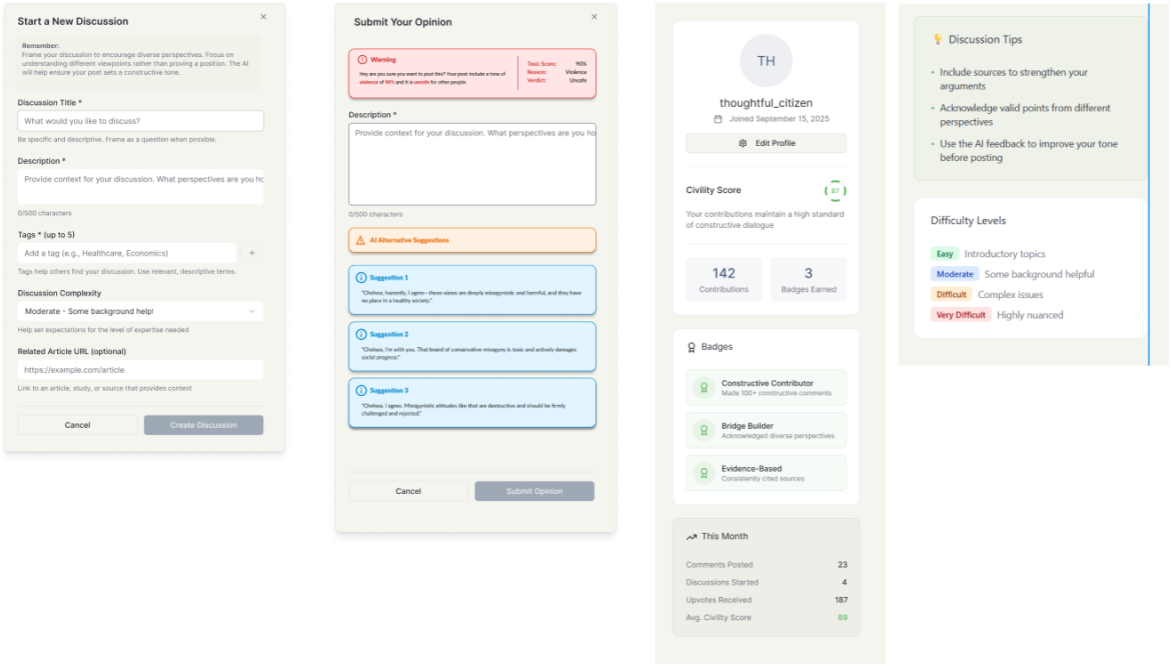


Figure 28 Screenshot of iteration 1 user interface designs, start discussion component, submit opinion component, user profile component and discussion tips component

Reflecting on the design, I notice the application lacks any enthusiasm and might have a target audience of people who might be highly educated and eloquently spoken which might make the moderation system impractical.

At that point I had to take a quick step back and look over my personas and remind myself what their needs and desires are for a platform. They expressed the need of a platform for debates and to potentially make friends, learn and have fun, without it being to serious and feel like a chore.

I navigated away from the sophisticated articles and decided to give the users free reign of what topics they'd like to discuss. I decided all discussions will be purely user generated for more freedom of choice. I decided to go back to the drawing board and reimagine the user experience from scratch.

4.1.1.3 Final Iteration

I have compared many typography styles and fonts; I wanted to create a collection that was easily legible yet still interesting and modern.

I chose a font called Danfo create by Afrotype in a navy shade for the logo as I thought it was an eye-catching and had a memorable design.

For the body and heading of the text I chose to settle on Roboto as it is a modern classic and easily legible. For the headers and buttons, I chose to capitalize the Roboto font to establish visual hierarchy.

Figure 29 Screenshot of a collection of typographies

I have chosen a colour palette for the typography/buttons/icons (main), viewpoint colours (secondary) and modals/extras (tertiary).

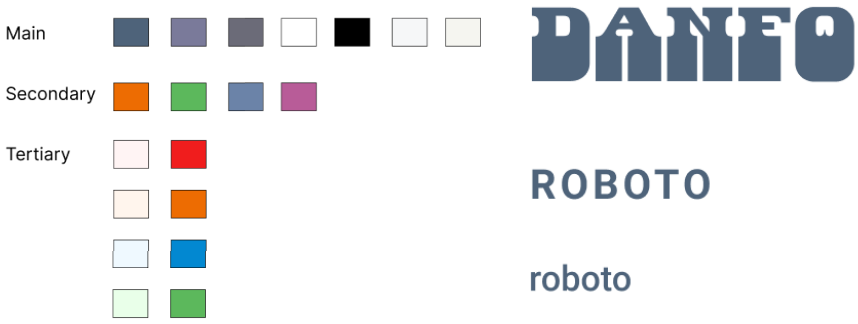


Figure 30 Screenshot of main, secondary and tertiary colour palettes, choice of typographies

Danfo - A typeface inspired by lettering on Lagos' Danfo buses. (n.d.). https://www.afrotype.com/danfo

I have created a word dictionary to help users understand the context and meaning behind the core phrases and features. This enhances the user experience by making the interface more intuitive and reducing potential confusion when navigating the platform.

Dictionary		
Phrase	Related Phrasings	Description
Common Ground	Consensus, Agreement, Middle Ground	A state of the discussion when after closing, majority of the discussion members are in the centre viewpoint.
Unresolved	Disagreement, No Consensus, Divided	A state of the discussion when after closing, majority of the discussion members are not in the centre viewpoint.
Discussions	Threads, Topics, Debates	Structured conversations on a specific topic involving multiple users.
Opinions	Comments, Posts, Responses, Arguments	Individual user shares their thoughts within a discussion.
Civility Score	Politeness Rating, Respectfulness Index	A numerical measure of how civil or respectful a discussion/opinion or user's participation is.
Viewpoint	Stance, Position, Perspective: For, Against, Centre, Undecided	The stance of a user or opinion in a discussion.
Moderation Scores	Safety Ratings, Toxicity Score	Automated scores indicating the appropriateness of user-generated content for each moderation label.
Verdict	Moderation Result, Safety Decision	Whether the user generated text is SAFE or UNSAFE after moderation.
AI Alternative Suggestions	Automated Suggestions, AI Edits	Alternative phrasings generated by Open-AI to improve civility.
Perspective Tags	Conservative, Liberal, Centrist, Progressive, Libertarian, Socialist, Green, Nationalist, Populist, Undecided	Labels indicating the political or ideological perspective

4.1.1.3.1 Login and Registration Page Designs

The login page has a simple, centred card layout with the "Common Ground" logo in the header, navigation links (Home, Login, Register), and a minimal form containing Email and Password fields with a Login button.

The register page has a more detailed form card with fields for Username, Title, Date of Birth, Bio, Location, Email, Password, Confirm Password, and Profile Picture (with an upload option).

Figure 31 Screenshot of final iteration; login, register and register with validation pages

The second variation includes inline validation error messages in red beneath each field (e.g., required field warnings, invalid email format, age restrictions, password mismatch). This highlights the error state UI, ensuring users are clearly guided to fix issues before submitting.

4.1.1.3.2 Discussion Board and Share Opinion Page Designs

The discussion board page displays a single discussion topic. "Girl Math - Is it a harmful term?" with a short description and a viewpoint distribution bar (For / Against / Centre / Undecided). Below, an Opinions section lists user opinions with profile pictures, username, timestamp, viewpoint label (colour-coded), opinion text, and Reply/View Replies links. A Share Opinion button with a countdown timer is visible in the opinion's header.

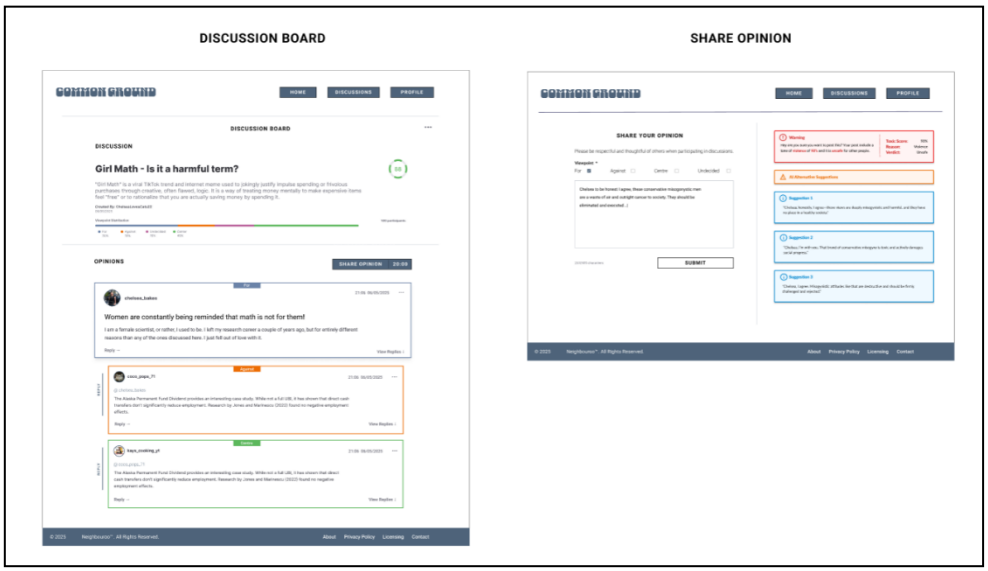


Figure 32 Screenshot of final iteration; discussion board and share opinion pages

The share opinion page is a form-based page where users submit their opinion and viewpoint on the discussion. It includes: A Viewpoint selector (For / Against / Centre / Undecided), A text area for writing the opinion and a submit button.

On the right side, an AI-powered panel provides real-time feedback including a warning box flagging potentially harmful or violent language, a toxicity score meter and an AI alternative suggestions section offering three reworded, more constructive versions of the user's opinion.

Confirmation Modal Component Designs

There are two versions of a confirmation modal triggered before submitting an opinion. The not revised modal is an orange-toned disclaimer modal warning that language is "unsafe," with blunt call to action buttons like Cancel and "Whatever, Submit." , signalling a dismissive tone to create the sense of accountability.

Figure 33 Screenshot of final iteration; not revised and revised confirmation modal variations

The revised modal is a green-toned Safe modal confirming the opinion contributes to healthy discourse, with standard *Cancel* and *Submit* buttons. The colour and tone shift reinforces positive reinforcement and pride in accountability.

4.1.1.3.3 Discussion Card Component Designs

This screenshot captures six different iterations and feature combinations of the same discussion. The base card includes a title, description, creator, date, positivity score, timer, participant count, and a "Request to Participate" button.

The common ground state includes a green-tinted card with "COMMON GROUND" stamp, indicating the discussion ended with an agreement. The unresolved state includes a red/pink-tinted card with "UNRESOLVED" stamp, indicating the discussion ended without agreement.

Figure 33 Screenshot of final iteration; iterations of discussion card designs

Topic tags were removed from the design to remove clutter and focus the importance on the title of the discussion.

4.1.1.3.4 All Discussion and Create Discussion Page Designs

The all discussions page is a listings page showing multiple discussion cards with a search bar and a new discussion button. Cards display title, description, stance tag, civility score and creator info.

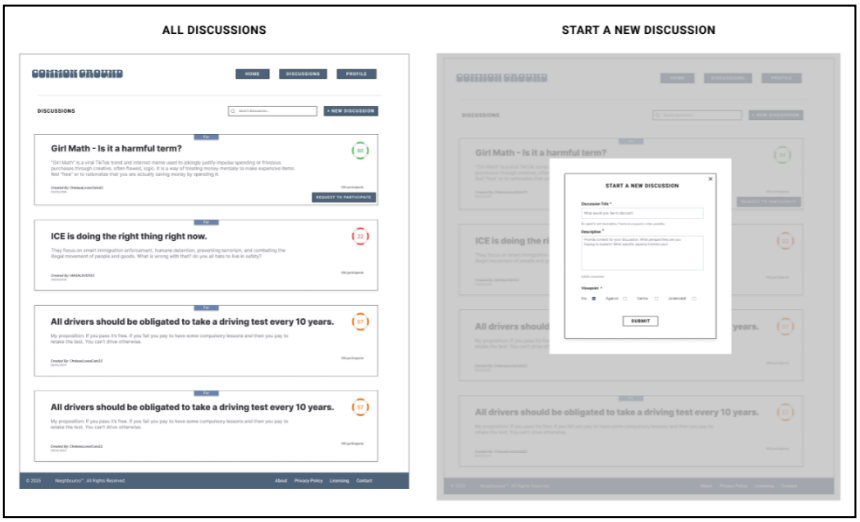


Figure 34 Screenshot of final iteration; all discussions and start new discussion pages

The start a new discussion page is a modal overlay with fields for discussion title, description, and viewpoint selector (For/Against/Centre/Undecided), and a Submit button. This makes the creation flow lightweight and not time consuming.

4.1.1.3.5 Profile Page Design

The profile page is a clean personal profile page displaying the users profile picture username, join date, DOB, and location. A Civility Score is displayed, reflecting the user's discourse quality on the platform. A short bio and an Edit Profile button are included.

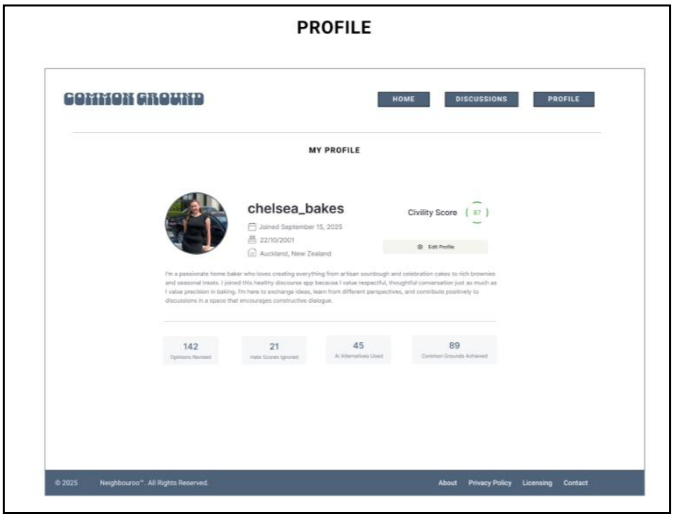


Figure 35 Screenshot of final iteration; user profile page

There are four components reflecting the users' statistics like Opinions Revised, Hate Scores Ignored, AI Alternatives Used, Common Grounds Achieved, which reinforce the platform's focus on healthy, constructive dialogue.

4.1.1.3.6 Conclusion

The designs presented for Common Ground cover the full user journey from registration and login to discussion participation and profile management. The wireframes maintain consistent branding, set layouts and a clear focus on constructive discourse throughout.

The visual and structural direction was briefly inspired by competitors' analysis from the requirements chapter, that exposed significant gaps in moderation and civility.

Common Ground aimed to address these shortcomings through deliberate design choices, guided by the platform's functional requirements including AI-powered harm detection, viewpoint selection, civility scoring, and discussion state indicators, all of which are clearly reflected across the designs.

The personas developed during the research phase further ensured that these decisions were grounded in real user needs, highlighting a balance between accessibility and meaningful engagement.

It is worth noting that the final UX/UI of the developed platform does not entirely reflect these designs and instead serve as the main conceptual reference point that informed and influenced decision-making throughout the development process as implementation progressed.

4.1.2 Process Design

The High-resolution versions of the diagrams can be found on Miro.
https://miro.com/app/board/uXjVJoNoh0o=/

4.1.2.1 System Architecture

The interface is built with React, leveraging its component-based architecture for UI reusability and its Virtual DOM for efficient, high-performance rendering. Vite serves as the build tool and development server, providing near-instant hot module replacement and optimized production bundles. Styling is handled through a utility-first approach using Tailwind CSS integrated with app-level CSS Variables for centralized theme and colour management.

The backend is powered by Flask, a lightweight Python framework used to build RESTful APIs and handle HTTP logic. PostgreSQL serves as the primary relational database, chosen for its reliability with write-heavy workloads and complex querying. During development, TablePlus is used for database management, and Insomnia is used for API testing and debugging.

The backend integrates my fine-tuned XLM-RoBERTa model to perform deep semantic analysis for harmful speech detection. For generating constructive alternative phrasings, the system utilizes the OpenAI GPT API, using its advanced reasoning capabilities for nuanced language generation.

Finally, I use Docker for containerized deployment so the project and all dependencies can be packaged and tested after submission, Render to host the backend, Firebase to host the frontend, Git for version control to manage code changes, and GitHub as the cloud repository for storing the code and submitting the project.

The entire environment is containerized using Docker to ensure consistency across development and submission. The project uses Git for version control with GitHub as the remote repository. The Flask backend is hosted on Render, while the React frontend is hosted on Firebase.

4.1.2.2 System Architecture Diagram

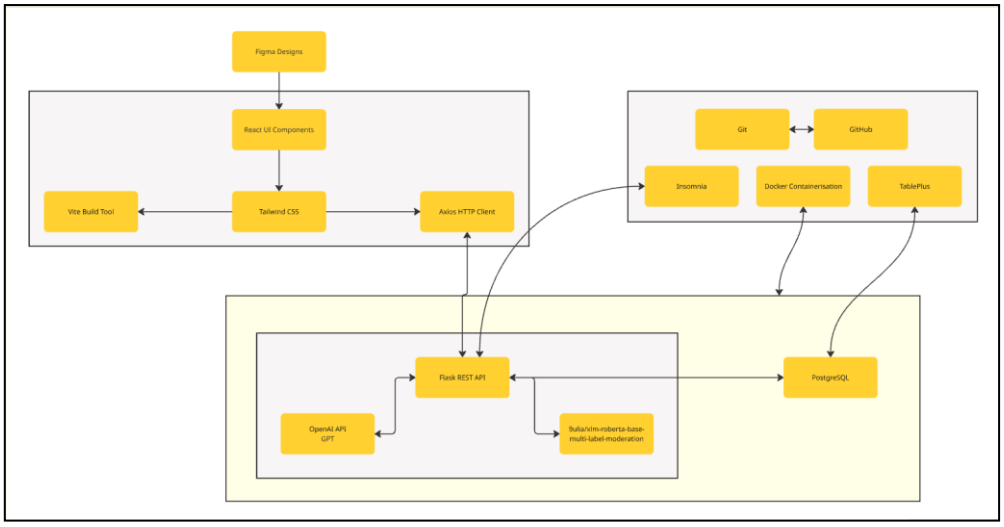


Figure 36 system architecture diagram

The diagram represents a full-stack system architecture and data lifecycle across the frontend, backend, and AI services:

1. Frontend: User triggers an action in the React interface.
2. Transport: Axios handles the asynchronous request to the backend.
3. Processing: The Flask API acts as the central hub, coordinating OpenAI, fine-tuned XLM-RoBERTa and PostgreSQL.
4. Output: Flask delivers the processed data back to the frontend to update the UI and provide user feedback.

To define the Process Design, a sequence diagram was created to details and the execution flow of a user request. This moves beyond the system architecture diagram to show the precise chronological order of operations, including how the system handles authentication, triggers the AI moderation pipeline, and ensures data integrity through PostgreSQL before the frontend reflects the change.

4.1.2.3 Sequence Diagram

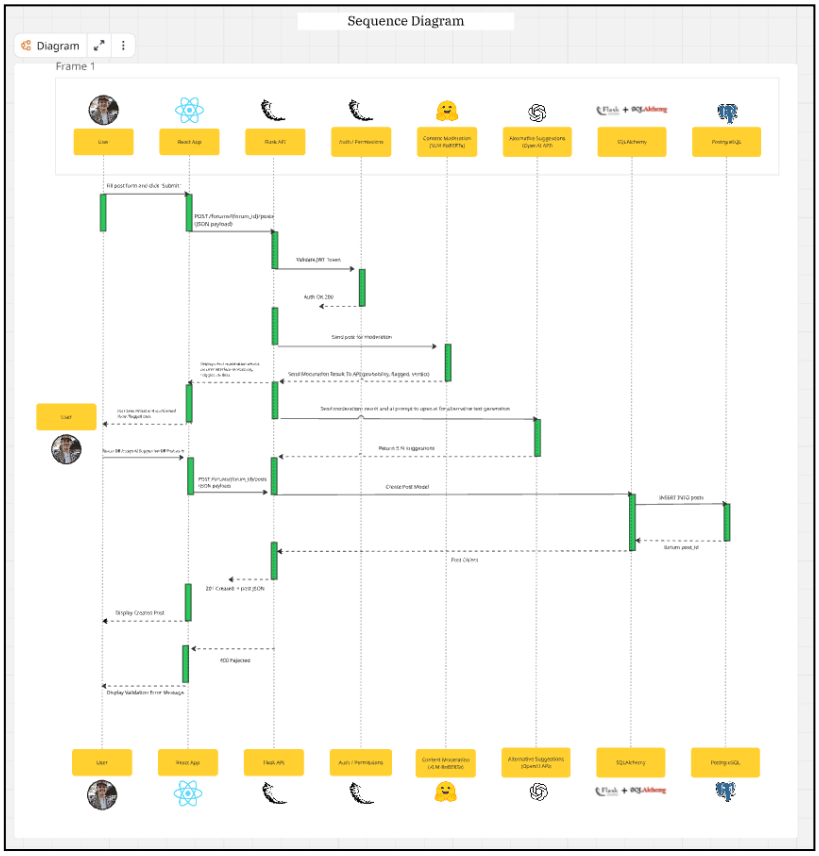


Figure 37 sequence diagram

Case Scenario: Logged in user creates a discussion.

The process begins when the User submits content through the React front-end. This triggers an asynchronous POST request containing a JSON payload sent to the Flask API. Before any data processing happens, the backend communicates with the Auth service to validate the user's JWT token. A successful "Auth OK" response allows the process to continue.

Once authorized, the Flask API forwards the content to the fine-tuned XLM-RoBERTa model. This service performs a deep semantic scan to detect harmful language. The moderation results are sent back to the API.

If the moderation service flags the content, the Flask API sends a request with the moderation report to the OpenAI API. The AI processes the context and returns constructive suggestions for alternative phrasing to help the user revise their text included when creating the discussion.

After the AI pipeline is complete, the Flask API passes the final post model to SQLAlchemy. The ORM executes an INSERT command to record the data in the PostgreSQL database. Once the database confirms the transaction with a success message, the backend generates a final response and returns a 201 Created status with the JSON payload to the React front-end.

To further clarify the moderation workflow, this diagram visualizes how the system processes and validates discussion text before presenting the refined results to the user.

Moderation Pipeline Diagram

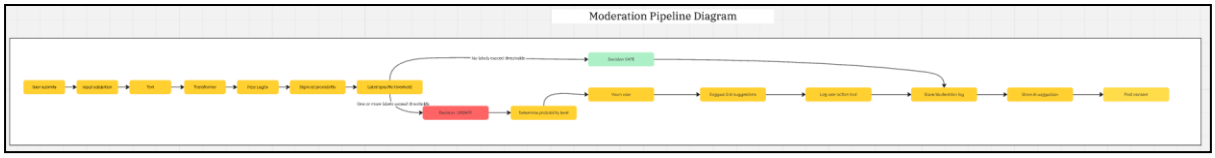


Figure 38 moderation pipeline diagram

The label specific threshold determines if text is safe based on moderation scores from the XLM-RoBERTa model for different categories, like hate speech, violence, sexual content etc. If any score is higher than the threshold limit for that category, the system sends the moderation results alongside a prompt to the OpenAI API to be analysed and receive back safer alternative suggestions based on the submitted text.

4.1.3 Database Design

4.1.3.1 Entity Relationship Diagrams

4.1.3.1.1 Iteration 1

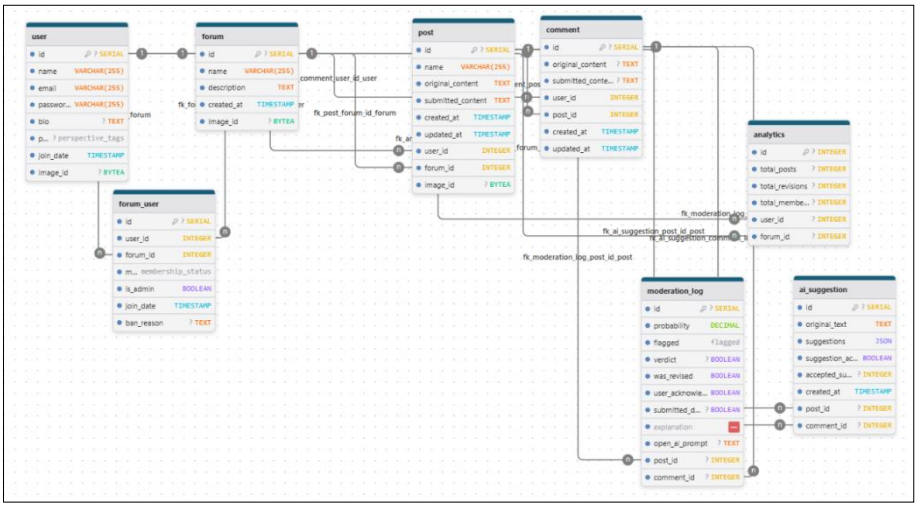


Figure 39 Entity Relationship Diagram 1

This is the first iteration of the entity relationship diagram. The tables included user, forum, forum_user, post, comment, moderation_log, ai_suggestions and analytics. The relationships weren't robust and were quite confusing.

The table names were revised to remove the generic naming conventions and instead apply the word dictionary. This was done to make the database schema more intuitive for the development process and align with the user interface design.

4.1.3.1.2 Iteration 2

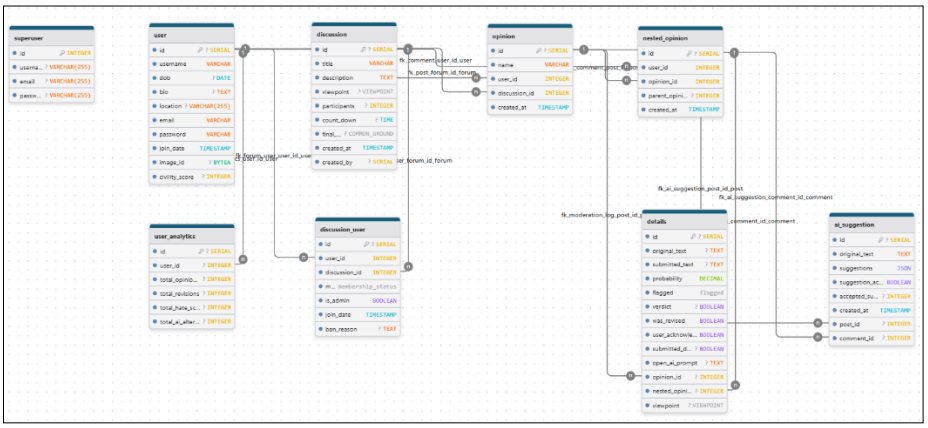


Figure 40 Entity Relationship Diagram 2

In the second iteration, the word dictionary was applied to the table names to match the naming conversion with the user interface design. A superuser table was added as an overall administration entity and the analytics table had some changes made to what data will be stored.

4.1.3.1.3 Final Iteration

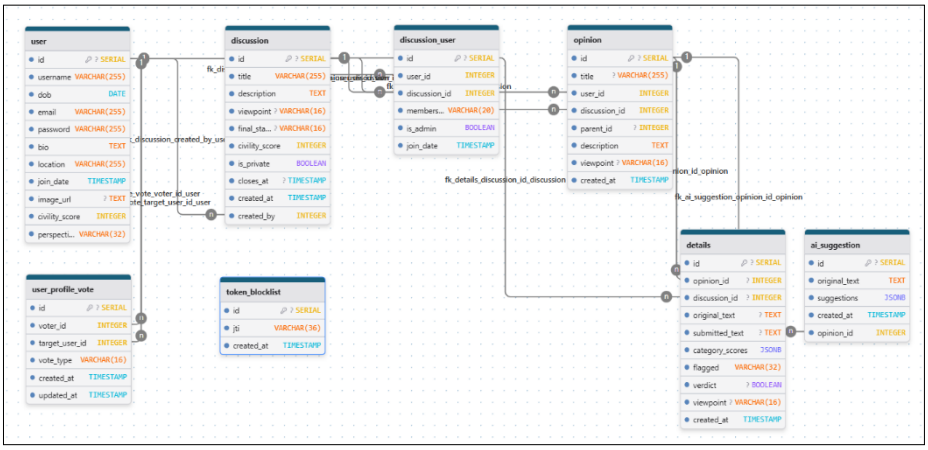


Figure 41 Final Entity Relationship Diagram

The final iteration of the entity relationship diagram was refined towards the end of the development cycle. Due to the time crunch some core changes were made to reduce the complexity of the schema. The superuser table was removed as it wasn't necessary. The token_blocklist table was added to block certain JWT token when they were expired or when the user logged out. The user analytics table was removed and replaced by the user_profile_vote table.

The nested_opinion table was removed, instead the opinion table gained a parent_id row for self-referencing to implement threaded conversations without repeating logic. The was_revised and submitted_despite_score rows were deleted as the logic to implement the features to trigger these booleans proved to be tedious to implement inside the front end.

4.1.3.2 Database Schema

Enum: perspective_tag
conservative
liberal
centrist
progressive
libertarian
socialist
green
nationalist
populist
undecided

Enum: viewpoint
for
against
centre
undecided

Enum: common_ground
achieved
unresolved
incomplete

Enum: membership_status
accepted
pending
rejected
not_a_member
blocked

Enum: flagged
unflagged
sexual_content
hate_speech
violence
harassment
self_harm
minors_exploitation
hate_speech_severe
violence_severe

Table: user		
Column Name	Data Type	Description
id	SERIAL (PK)	Unique user ID
username	VARCHAR(255)	User's unique username
dob	DATE	Date of birth
email	VARCHAR(255)	User's unique email address

password	VARCHAR(255)	User's password (hashed)
bio	TEXT	User's biography
location	VARCHAR(255)	User's location
join_date	TIMESTAMP	Date the user joined
image_url	TEXT	Profile image URL
civility_score	INTEGER	Civility score (default 100)
perspective_tag	ENUM perspective_tag	User's political perspective

Table: discussion		
Column Name	Data Type	Description
id	SERIAL (PK)	Unique discussion ID
title	VARCHAR(255)	Discussion title
description	TEXT	Discussion description
viewpoint	ENUM viewpoint	Discussion viewpoint (enum: for, against, etc.)
final_status	ENUM common_ground	Final status (enum: achieved, unresolved, incomplete)
civility_score	INTEGER	Civility score (default 50)
is_private	BOOLEAN	Whether the discussion is private or public.
closes_at	TIMESTAMP	Optional closing date
created_at	TIMESTAMP	Creation date
created_by	INTEGER (FK)	User who created the discussion

Table: discussion_user		
Column Name	Data Type	Description
id	SERIAL (PK)	Unique ID
user_id	INTEGER (FK)	Linked user
discussion_id	INTEGER (FK)	Linked discussion
membership_status	ENUM membership_status	Membership status (enum: accepted, pending, etc.)
is_admin	BOOLEAN	Whether user is admin in discussion
join_date	TIMESTAMP	Date joined

Table: opinion		
Column Name	Data Type	Description
id	SERIAL (PK)	Unique opinion ID
title	VARCHAR(255)	Opinion title
user_id	INTEGER (FK)	Authoring user
discussion_id	INTEGER (FK)	Related discussion
parent_id	INTEGER (FK)	Parent opinion (for threads)
description	TEXT	Opinion text
viewpoint	ENUM viewpoint	Opinion viewpoint (enum)
created_at	TIMESTAMP	Creation date

Table: details		
Column Name	Data Type	Description
id	SERIAL (PK)	Unique details ID
opinion_id	INTEGER (FK)	Related opinion
discussion_id	INTEGER (FK)	Related discussion
original_text	TEXT	Original text
submitted_text	TEXT	Submitted text
category_scores	JSON	Category scores (JSON object)
flagged	flagged	Moderation flag (enum: unflagged, hate_speech, etc.)
verdict	BOOLEAN	Final verdict (SAFE/UNSAFE)
viewpoint	ENUM viewpoint	User's viewpoint (enum)
created_at	TIMESTAMP	Creation date

Table: ai_suggestion		
Column Name	Data Type	Description
id	SERIAL (PK)	Unique suggestion ID
original_text	TEXT	Original text to improve
suggestions	JSON	List of AI suggestions
created_at	TIMESTAMP	Creation date
opinion_id	INTEGER (FK)	Related opinion

Table: user_profile_vote		
Column Name	Data Type	Description
id	SERIAL (PK)	Unique vote ID
voter_id	INTEGER (FK)	User casting the vote
target_user_id	INTEGER (FK)	User receiving the vote
vote_type	VARCHAR(16)	upvote or downvote
created_at	TIMESTAMP	Creation date
updated_at	TIMESTAMP	Last update date

Table: token_blocklist		
Column Name	Data Type	Description
id	SERIAL (PK)	Unique blocklist ID
jti	VARCHAR(36)	JWT token identifier
created_at	TIMESTAMP	Date token was blocked

4.1.3.3

4.1.3.4 Table Relationships

Relationships			
Table	Relationship	Table	Notes
user	1-to-many	discussion	via discussion.created_by
user	many-to-many	discussion	via discussion_user
user	1-to-many	opinion	
discussion	1-to-many	opinion	

opinion	self-referential 1-to-many	opinion	via parent_id
opinion	1-to-many	details	
discussion	1-to-many	details	
opinion	1-to-many	ai_suggestion	
user	self-referential many-to-many	user	via user_profile_vote as voter and target
token_blocklist	none		

5 Implementation

5.1 Overview

This chapter explains the development process of the platform, describing the technical decisions made during implementation, the challenges I encountered, and the solutions I applied. The application was built across a series of sprints following an Agile SCRUM methodology, progressing from model training and backend infrastructure through to frontend development and final deployment. The following sections address each major component of the system which are: the machine learning moderation pipeline, the backend REST API, and the React-based frontend.

GitHub Copilot was used as a coding assistant throughout development, primarily for generating realistic seed data, repetitive endpoint structures, and test data generation. All Copilot-generated code was revised, changed, and integrated my me to ensure it met the project's specific requirements. The OpenAI API was used as an external service within the application itself, not as a development tool but for generating alliterative AI suggestions for user generated text in the front-end.

5.2 Machine Learning

5.2.1 Development Environment and Tools

The machine learning pipeline was developed and trained in Google Colab using a Jupyter Notebook, which provided access to GPU through Google's cloud infrastructure. This was a necessary given the computational needs of fine-tuning a large transformer model that my laptops CPU could not handle.

The training, validation, and testing pipeline relied on several Python libraries. Pandas and NumPy handled data manipulation, including class distribution analysis, column renaming, and array operations used in the compute_metrics and build_labels functions. PyTorch was used for tensor operations, GPU acceleration, and defining the BCEWithLogitsLoss function within the custom WeightedTrainer.

Matplotlib and Seaborn were used to visualise label distributions and confusion matrices. The Hugging Face imported the transformers library to load the AutoTokenizer, AutoModelForSequenceClassification, and TrainingArguments, while the datasets library handled loading, splitting, and managing the dataset across training, validation, and test splits.

Finally, scikit-learn provided the train_test_split function for stratified dataset splitting, and its metrics module was used to compute F1 scores, confusion matrices, and the full classification report across all eight labels.

5.2.2 Dataset Choice

The fine-tuning of the XLM-RoBERTa model required a multilingual and multilabel of pre-labels harmful speech examples. The KoalaAI Text-Moderation-Multilingual dataset was selected for this purpose, sourced from Hugging Face. The dataset contains 1.7 million rows of user-generated content, across 17 languages, including English, German, French, Spanish, Italian, Swedish, Finnish, Polish, Czech, Latvian, Chinese, Japanese, Korean, Russian, Ukrainian, Belarusian and Kazakh.

https://huggingface.co/datasets/KoalaAI/Text-Moderation-Multilingual

Figure 42 Screenshot of KoalaAI Text-Moderation-Multilingual dataset card

The dataset has 9 distinguishable labels and 8 of them being toxicity related binary entries used for training, validation and testing: sexual content, hate speech, violence, harassment, self-harm, minors' exploitation, severe hate speech, and severe violence. These aligned directly with the platform's moderation requirements.

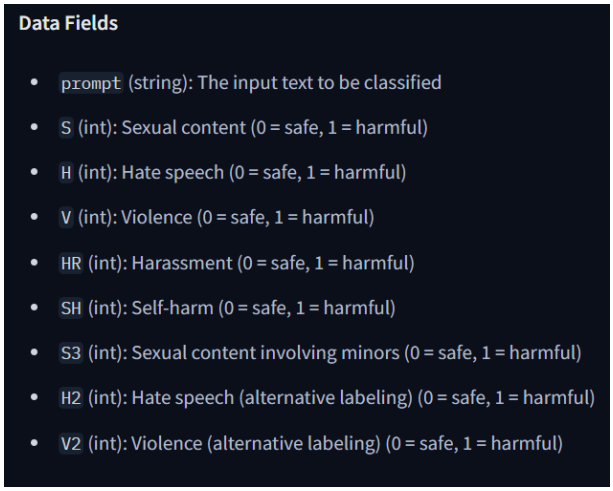


Figure 43 Screenshot of data fields

5.2.3 Machine Learning Moderation Pipeline

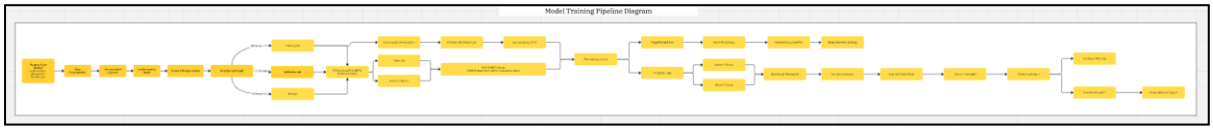


Figure 44 machine learning moderation pipeline diagram

This is a Model Training Pipeline Diagram showing how a moderation model is built and evaluated.

Flow:

1. Input Data → Preprocessing → Split data → Feature engineering → Training setup
2. Branching paths for different model/config options feeding into a central Training Process
3. Post-training: Evaluate model → Validate results → Benchmark testing → Store model
4. Deployment check:
    - Passes → Log results → Send to API → Final Model deployed
    - Fails → Retrain loop → back to training
5. Final trained and validated model ready for use

5.2.4 Model Training, Validation and Testing Pipeline Diagram

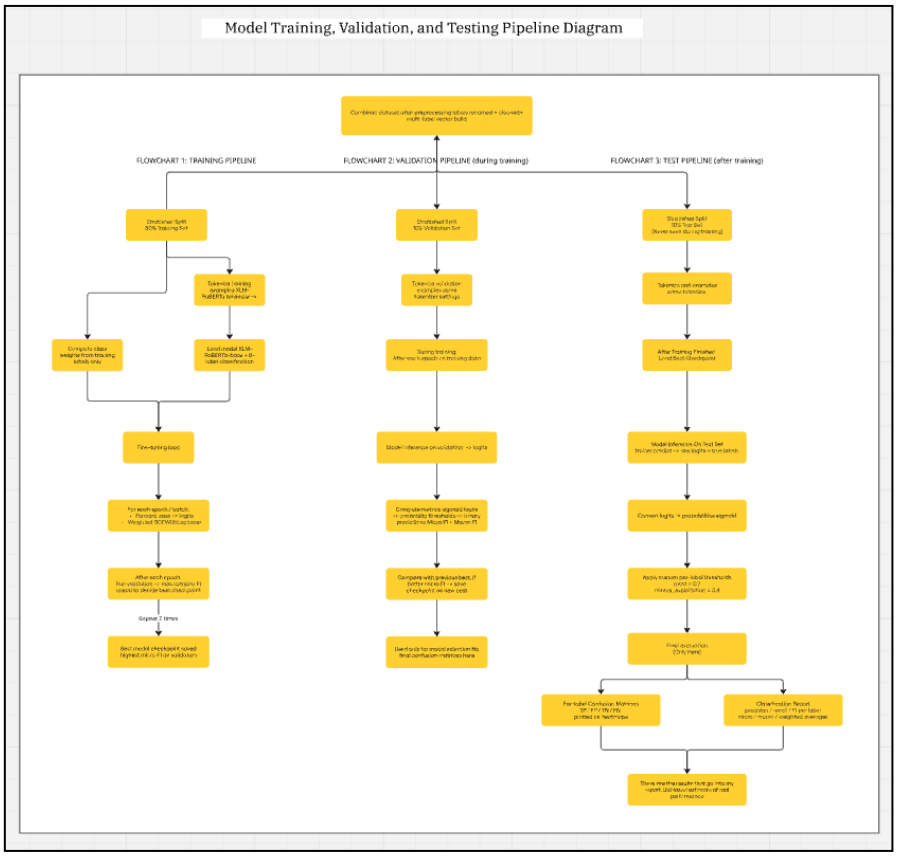


Figure 45 model training, validation and testing pipeline diagram

This is a Model Training, Validation, and Testing Pipeline Diagram for XLM-RoBERTa Multi-label Classification split into 3 pipelines.

Flowchart 1: Training Pipeline (80% of data):

1. Stratified split → Tokenize with XLM-RoBERTa → Load model (8-label classification)
2. Compute class weights → Fine-tuning loop → Forward pass → logits → Weighted BCEWithLogitsLoss
3. After each epoch: run validation → track best checkpoint
4. Repeat 3 times → Save best model checkpoint

Flowchart 2: Validation Pipeline (10% of data, during training):

1. Stratified split → Tokenize → Run inference after each epoch
2. Compute Micro F1 + Macro F1 → Compare with previous best → Save new best checkpoint if improved
3. Used only for model selection

Flowchart 3: Test Pipeline (10% of data, after training):

1. Never-seen data → Tokenize → Load best checkpoint
2. Run inference → Convert logits to probabilities (sigmoid)
3. Apply custom per-label thresholds (most=0.7, minors_exploitation=0.4)
4. Final evaluation → Per-label confusion matrices (heatmaps) + Classification report (precision/recall/F1)
5. Results go into the unbiased real-performance report

5.2.5 Preprocessing

During data manipulation and preprocessing I had to analyse of the distribution of the labels. The dataset had a large number of harassment example and a very small number of self-harm, severe hate speech, severe violence and exploitation of minor examples. This analysis revealed a significant class imbalance: harassment accounted for the largest share of flagged examples, while minority classes such as severe hate speech, severe violence, and minors' exploitation had substantially fewer positive entries.

I was worried that the fine-tuned model might not be as accurate when predicting minority classes and put more emphasis on the majority classes with a larger number of examples. I thought about balancing the dataset which would equalise the number of entries per label across all labels but soon realised that it would not be a realistic environment as there are many more instances of harassment related user generated content over the severe categories in the real-world distribution of these examples present on social media platforms.

Instead, class imbalance was dealt with through a custom WeightedTrainer class that applied inverse-frequency class weights to the binary cross-entropy loss function BCEWithLogitsLoss, causing the model to penalise misclassifications of minority classes more heavily during training.

Contributors, P. (2023, January 1). BCEWithLogitsLoss.
https://docs.pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html

5.2.6 Training and Validation

The dataset was split into 80% training, 10% validation, and 10% for testing. Stratification made sure that there was a proportional distribution of labels was preserved across all three splits.

```python
from sklearn.model_selection import train_test_split
X = list(dataset["data"]["prompt"])
y = np.array(dataset["data"]["labels"])

strat = y.sum(axis=1)

X_train, X_temp, y_train, y_temp = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=42,
    stratify=strat
)

strat_temp = y_temp.sum(axis=1)

X_val, X_test, y_val, y_test = train_test_split(
    X_temp,
    y_temp,
    test_size=0.5,
    random_state=42,
    stratify=strat_temp
)
```

Figure 46 code snippet of training, validation and test split

The 80-10-10 ratio is a widely accepted practice across machine learning. A large training set (80%) allows the model to learn patterns and relationships from the data in detail. The validation set (10%) is used during training for hyperparameter tuning and to prevent overfitting. Finally, the test set (10%) provides an unbiased evaluation of the model's performance on completely unseen data.

Splitting training data into Train, Validation, and Test Sets. (n.d.). Pecan Help Center.
https://help.pecan.ai/en/articles/6454518-splitting-training-data-into-train-validation-and-test-sets

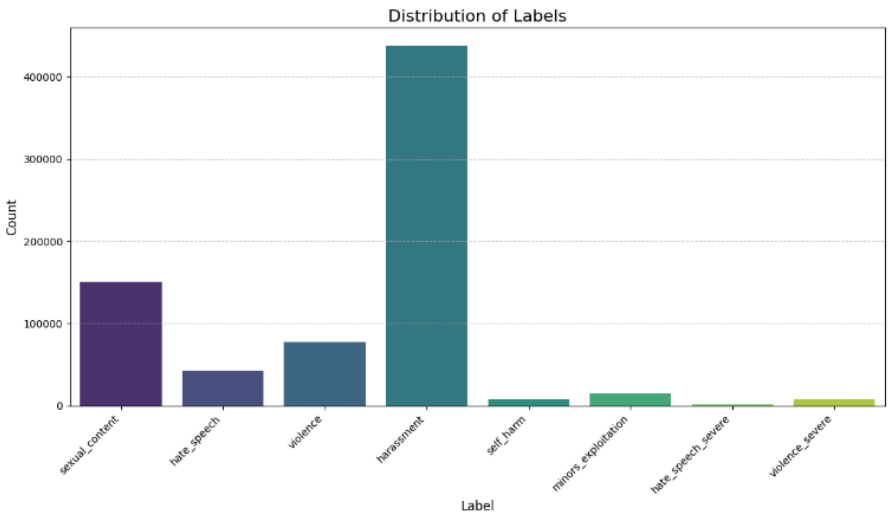


Figure 47 bar chart of the distribution of labels in the dataset

I then checked the proportion distribution of the positive labels which are the entries that are flagged as positive within the categories. These entries would include examples of actual words or phrases that would flag the content moderation system, so they are crucial for the training process to determine the accuracy of real-world examples.

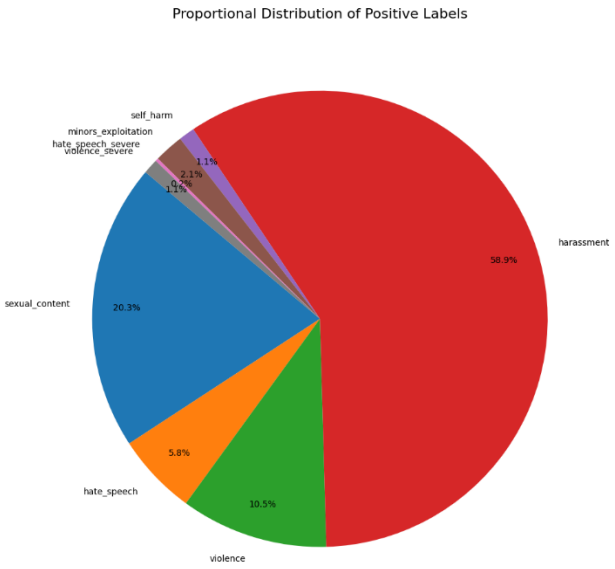


Figure 48 pie chart of the proportional distribution of positive labels in the dataset

Tokenization was done using Hugging Face's AutoTokenizer loaded from the XLM-RoBERTA -base checkpoint. The SentencePiece tokenizer used by XLM-RoBERTa handles multilingual character sets, which was important given the diversity of languages in the training data.

The pre-trained XLM-RoBERTa-base model was loaded using Hugging Face's AutoModelForSequenceClassification with the number of output labels set to eight, matching the eight toxicity categories. A multi-label classification head was used, with sigmoid activation applied to each output independently, this allowed the model to flag multiple categories simultaneously for a single piece of text. For example, a text that contains both harassment and violence.

```
model_checkpoint = "xlm-roberta-base"
num_labels = 8

tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)

model = AutoModelForSequenceClassification.from_pretrained(
    model_checkpoint,
    num_labels=num_labels,
    problem_type="multi_label_classification"
)
```

Figure 49 code snippet of model and tokenizer importing code

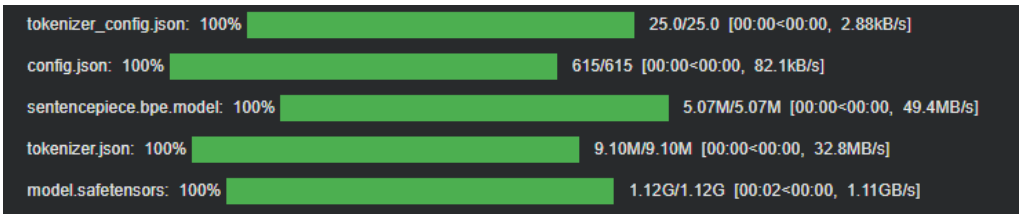


Figure 50 code snippet of the loaded model and tokenizer

Training was iterated over three epochs using the Hugging Face Trainer API, customised as WeightedTrainer to implement the weighted loss function. Training arguments were set to log loss and evaluate on the validation set at the end of each epoch. The model was trained on Google Colab's A100 GPU. The model took approximately 23425.58 seconds to train, which is about 6 hours and 30 minutes.

```
from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="./results",
    eval_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    gradient_accumulation_steps=2,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_steps=100,
    load_best_model_at_end=True,
    metric_for_best_model="micro_f1",
    greater_is_better=True,
    fp16=torch.cuda.is_available(),
    report_to="none"
)

trainer = WeightedTrainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    processing_class=tokenizer,
    compute_metrics=compute_metrics,
)

trainer.train()
```

Figure 51 code snippet of training arguments and weight trainer

The training done using the TrainingArguments class from the Transformers library. These parameters control how the model learns and how its performance is evaluated during training.

A low learning rate of 2e-5 was used to make sure that the model updates its weights as training goes on. This is important when fine-tuning a pre-trained transformer model, because large updates at the beginning could damage the knowledge it has already learned.

The model was trained for three epochs, meaning it passed through the entire dataset three times. This provides a balance between allowing the model to learn the task and avoiding overfitting.

A batch size of 8 was used for both training and evaluation. Since hardware memory is limited, gradient accumulation was applied over two steps to simulate a larger effective batch size of 16. This improves training stability without requiring additional resources.

To reduce overfitting, weight decay of 0.01 was applied as a form of regularisation.

Opinion: Training Argument Fine Tuning MLM RoBERTA - Intermediate - Hugging Face Forums. (2025, January 9). Hugging Face Forums. https://discuss.huggingface.co/t/opinion-training-argument-fine-tuning-mlm-roberta/135013

The model was evaluated at the end of each epoch, and the best-performing version was saved. Performance was measured using the micro F1 score, which is suitable for multi-label classification tasks and provides a better indication of overall performance than accuracy when classes are imbalanced just like in my case.

Mixed precision (fp16) was activated when the A100 GPU was available, improving training speed and reducing memory usage.

Finally, logging was performed every 100 steps to monitor progress.

Epoch	Training Loss	Validation Loss	Micro F1	Macro F1
1	0.501300	0.323397	0.747230	0.623185
2	0.249400	0.265895	0.834133	0.776317
3	0.299000	0.265846	0.867901	0.838966

Training loss decreased after each epoch, from 0.501 in epoch one to 0.249 in epoch two, before a slight increase to 0.299 in the third epoch. This minor increase in training loss alongside a continuing reduction in validation loss (from 0.323 to 0.266) proved that the model was not overfitting and was learning, adapting and predicting well on unseen data. The macro F1-score improved from 0.623 after the first epoch to 0.839 after the third, proving that each epoch of training it improvement in classification quality.

Metric	Value
global_step	243225
training_loss	0.343454366868833
train_runtime	23425.5785
train_samples_per_second	166.126
train_steps_per_second	10.383
total_flos	1.6948888923083328e+17
epoch	3.0

5.2.7 Model Deployment

The trained model was then pushed to Hugging Face Hub under the identifier 9ulia/xlm-roberta-base-multi-label-moderation and imported directly into the Flask backend through the Hugging Face transformers library.

https://huggingface.co/9ulia/xlm-roberta-base-multi-label-moderation

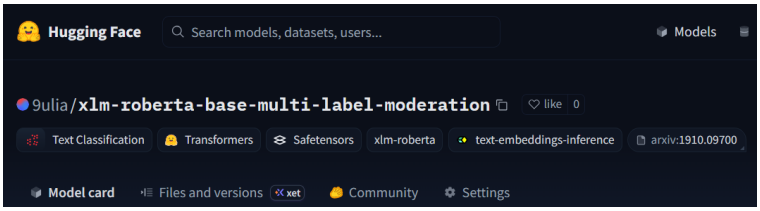


Figure 52 screenshot of the model being deployed to huggingface

5.3 Backend API And Moderation Pipeline Integration

The backend was implemented as a RESTful API using Flask, a lightweight Python web framework. The application follows a Model-View-Controller pattern, with blueprints used to organise routes by resource type, SQLAlchemy ORM models representing the database schema, and service modules for moderation scoring and JWT token management.

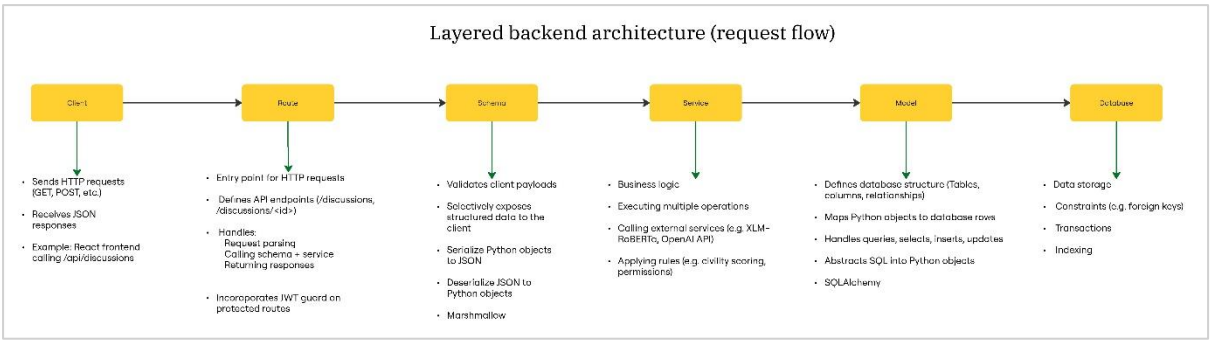


Figure 53 diagram of backend architecture

I chose PostgreSQL for the database because it's robust, supports complex relational schemas. The schema supports threaded discussions through a self-referential foreign key on the opinion table (parent_id), allowing replies to be nested deep.

SQLAlchemy manages how data is stored and queried in a database. Marshmallow manages how that data is converted into formats like JSON for APIs. For example: A user sends JSON data to the API. Marshmallow validates the JSON and converts it into a Python dictionary or object which is known as deserialization. SQLAlchemy then takes that object and saves it to the database as a new row. I then query the database to get that data back as a Python object. Finally, Marshmallow converts that complex object back into JSON to send back to the user which is also known as serialization.

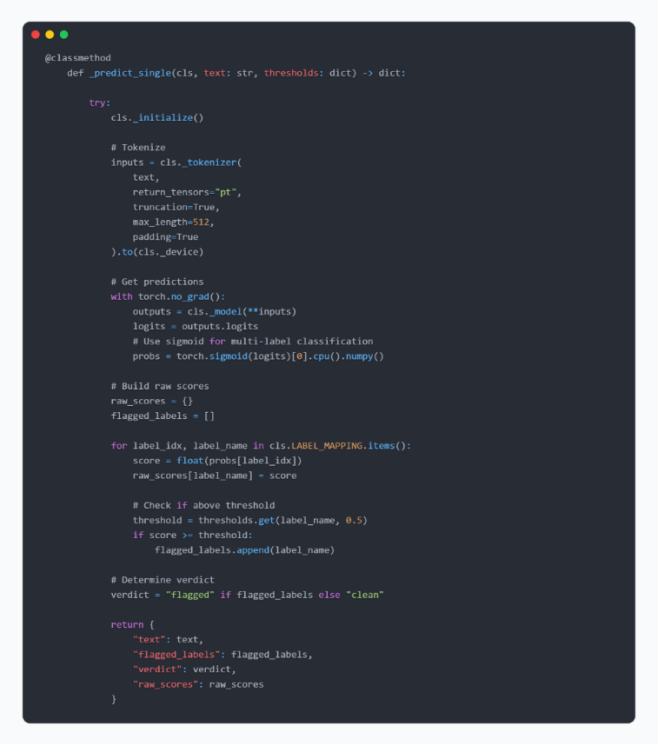
Lopez, T. (2023, December 1)

The model was then integrated into the backend as a moderation service that was triggered whenever a user submits an opinion or discussion title/description text. The service accepts the raw input text, tokenizes it, runs an inference pass through the fine-tuned model, and returns a dictionary of category scores using sigmoid-activated probabilities.

```python
class HFService:
    """Service for Hugging Face model-based moderation"""

    MODEL_NAME = "9ulia/xlm-roberta-base-multi-label-moderation"
```

Figure 54 code snippet of the model being loaded into the backend



```python
@classmethod
    def _predict_single(cls, text: str, thresholds: dict) -> dict:

        try:
            cls._initialize()

            # Tokenize
            inputs = cls._tokenizer(
                text,
                return_tensors="pt",
                truncation=True,
                max_length=512,
                padding=True
            ).to(cls._device)

            # Get predictions
            with torch.no_grad():
                outputs = cls._model(**inputs)
                logits = outputs.logits
                # Use sigmoid for multi-label classification
                probs = torch.sigmoid(logits)[0].cpu().numpy()

            # Build raw scores
            raw_scores = {}
            flagged_labels = []

            for label_idx, label_name in cls.LABEL_MAPPING.items():
                score = float(probs[label_idx])
                raw_scores[label_name] = score

                # Check if above threshold
                threshold = thresholds.get(label_name, 0.5)
                if score >= threshold:
                    flagged_labels.append(label_name)

            # Determine verdict
            verdict = "flagged" if flagged_labels else "clean"

            return {
                "text": text,
                "flagged_labels": flagged_labels,
                "verdict": verdict,
                "raw_scores": raw_scores
            }
```

Figure 55 code snippet of the predict function

These scores are stored in the details table in the PostgreSQL database, linked to the corresponding opinion or discussion via foreign key relationships.

category_scores
{"sexual_content": 0.02, "hate_speech": 0.04, "violence": 0.1, "harassment": 0.06, "self_harm": 0.09, "minors_exploitation": 0.03, "hate…
{"sexual_content": 0.03, "hate_speech": 0.08, "violence": 0.05, "harassment": 0.14, "self_harm": 0.02, "minors_exploitation": 0.09, "hat…
{"sexual_content": 0.12, "hate_speech": 0.4, "violence": 0.08, "harassment": 0.07, "self_harm": 0.07, "minors_exploitation": 0.06, "hate…

Figure 56 screenshot of category_scores table and row entries

A score threshold of 0.5 was applied to determine whether a given label is considered flagged.

```python
DEFAULT_THRESHOLDS = {
    "sexual_content": 0.5,
    "hate_speech": 0.5,
    "violence": 0.5,
    "harassment": 0.5,
    "self_harm": 0.5,
    "minors_exploitation": 0.1,
    "hate_speech_severe": 0.1,
    "violence_severe": 0.1
}
```

Figure 57 code snippet of default thresholds

When one or more labels exceed this threshold, the verdict is marked as UNSAFE and the OpenAI API is triggered to generate three alternative phrasings of the user's original text.

These suggestions are designed to keep the user's intended meaning while changing and replacing the harmful elements and are stored in the ai_suggestion table for ready for GET requests made by the frontend.

```python
class OpenAIService:
    """Service for OpenAI API calls"""
    MODEL = "gpt-3.5-turbo"

    def __init__(self):
        api_key = os.getenv("OPENAI_API_KEY")
        if not api_key:
            raise ValueError("OPENAI_API_KEY environment variable not set")
        self.client = OpenAI(api_key=api_key)

    def generate_alternatives(self, original_text: str, tone: str = "neutral") -> dict:
        """
        Generate 3 alternative suggestions for the given text.
        """

        system_prompt = (
            "You are an assistant for a debate platform similar to Reddit, X and 4chan. "
            "Your goal is to help users rephrase their statements to avoid direct harm or hate speech.
"
            "Allow some ragebait, but ensure that the suggestions steer users towards more civil and
engaging discourse."
        )
        user_prompt = (
            f"Please generate exactly 3 alternative suggestions for the following text:\n\n"
            f"Original text: \"{original_text}\"\n"
            "Requirements:\n"
            "1. Keep the original meaning and intent\n"
            "2. Make the suggestions more constructive\n"
            "3. Always return 3 suggestions, one per line, without numbering or bullet points\n"
            "4. Do not include explanations, just the text suggestions"
        )
        try:
            response = self.client.chat.completions.create(
                model=self.MODEL,
                messages=[
                    {"role": "system", "content": system_prompt},
                    {"role": "user", "content": user_prompt}
                ],
                temperature=0.7,
                max_tokens=800,
            )
            response_text = response.choices[0].message.content
            suggestions = [s.strip() for s in response_text.split('\n') if s.strip()]
            return {
                "suggestions": suggestions[:3],
                "prompt_used": user_prompt,
                "model_used": self.MODEL,
                "tokens_used": {
                    "input_tokens": getattr(response.usage, "prompt_tokens", None),
                    "output_tokens": getattr(response.usage, "completion_tokens", None),
                    "total_tokens": getattr(response.usage, "total_tokens", None)
                }
            }
        except Exception as e:
            return {
                "suggestions": [],
                "error": str(e),
                "prompt_used": user_prompt,
                "model_used": self.MODEL,
                "tokens_used": None
            }
```

Figure 58 code snippet of OpenAIService class including; prompt engineering functions

ai_suggestion

id	original_text	suggestions
1	You can write tasting notes in cursive all you want, but if the drink tastes…	["You can romanticize bitterness all you want, but the drink still tastes li…

Figure 59 screenshot of ai_suggestion table and rows with JSON structured suggestion entries

The moderation pipeline was tested end-to-end using the browser's network inspector. It confirmed that from the user submission to moderation result, including model inference, database writes, and OpenAI API calls the pipeline worked smoothly. Further results can be found in the testing and evaluation chapter.

5.3.1 Civility Score

The civility score calculates the harm-risk of text on a 0–100 scale, with a higher score indicating more civil text. It is computed from the moderation category scores produced by the fine-tuned XLM-RoBERTa moderation and stored inside the details record.

Standard harmful categories include sexual_content, hate_speech, violence, harassment, self_harm and have the 1.0 weight. Severe categories include minors_exploitation, hate_speech_severe, violence_severe and use a 3.0 weight because I believe the

use of language that flag these categories should be penalised more. Each category score $s_i$is normalized and clamped to the range [0,1] A weighted risk score is then calculated as:

$$\text{risk} = \min\left(1, \sum_i \omega_i S_i\right)$$

$w_i$represents the weight assigned to category i and the civility score is calculated as $\text{civility} = 100 - \text{round}(\text{risk} \times 100)$. This means a risk of 0.82324 gives civility = 100 − round(82324) = 18. If the users text was not flagged, then the category values are treated as 0.0. This is how the detail civility score is calculated.

Figure 60 code block of CivilityService class; harmful weights object and civility score calculation functions per discussion, opinion and user

The civility of a discussion is calculated as the rounded average of all associated Details records, this includes the actual discussion detail record and the detail records of all the individual opinions posted inside it.

$$\text{civility}_{\text{discussion}} = \text{round}\left(\frac{1}{\text{n}}\sum_{\text{j=1}}^{\text{n}} \text{civility}_{\text{j}}\right)$$

Example

A discussion has 4 Details records with the following civility scores: $\text{civility}_1 = 82$, $\text{civility}_2 = 65$, $\text{civility}_3 = 90$ and $\text{civility}_4 = 73$. The total is then calculated: $82 + 65 + 90 + 73 = 310$. The sum is then divided by the number of details records and rounded to the nearest integer: $\frac{310}{4} = \text{round}\ 77.5 = 78$.

Finally, a user's civility score is defined as the average civility of all Details records associated with their opinions. Scores are always kept within the range 0 to 100 and rounded to whole numbers. The total risk value is limited to a maximum of 1.0 to ensure it does not grow beyond the intended scale.

The civility score is designed to be simple to compute and interpretable while still reflecting differences in severity across different types of harmful content. A weighted sum of normalized category scores allows multiple forms of harm to contribute as a collection, rather than relying on a single category as the moderation pipeline was it was initially designed to do. Assigning higher weights to severe categories makes sure that extreme harm has a greater impact on the final score. The use of a cap on risk prevents extreme inputs from skewing results so when a score hits 1 or above that it's also treated as 1.

Example

hate_speech = 0.6 (weight 1.0) , violence = 0.5 (weight 1.0) , hate_speech_severe = 0.4 (weight 3.0)

$$(1.0 \cdot 0.6) + (1.0 \cdot 0.5) + (3.0 \cdot 0.4) = 0.6 + 0.5 + 1.2 = 2.3$$

The raw risk = 2.3, which is outside the intended range, so the risk is 1, which is 100% risk which means the civility is 0. civility=100−round(risk×100) is the same as 100−100=0.

5.3.2 Authentication and Security

User authentication was implemented using JSON Web Tokens through the flask-jwt-extended library. After successful login, users receive an access token which must be included in the Authorization header of all protected requests. Route protection is created using the @jwt_required() decorator, which rejects requests with missing or invalid tokens with a 401 status code.

```python
@auth_bp.route('/me', methods=['GET'])
@jwt_required()
def get_me():
    """Return the full profile of the currently authenticated user."""
    identity = get_jwt_identity()

    user = User.query.get(int(identity))
    if not user:
        return jsonify({'error': 'User not found'}), 404

    return jsonify(user.to_dict(include_private=True, viewer_id=int(identity), include_votes=True)),
200
```

Figure 61 code snippet of "me" authenticated route

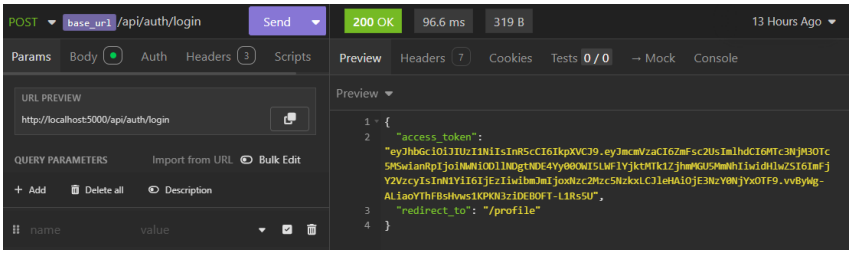


Figure 62 screenshot of insomnia; backend responds with a JWT access token to authenticated user

Passwords are securely hashed using PBKDF2 with SHA-256 before storage, implemented through Werkzeug's generate function. This ensures plaintext passwords are never stored in the database,

protecting against potential data breaches. Password verification uses check function to securely compare during authentication.

```
from werkzeug.security import generate_password_hash, check_password_hash

    hashed_password = generate_password_hash(password)
```

Figure 63 code snippet of werzeug generate_password_hash function

A token_blocklist table was created to support logout and session invalidation: after logout, the token's unique identifier (JTI) is written to this table, and all incoming tokens are checked against it before being accepted. This prevents token reuse after a user has logged out, which is a common security vulnerability in JWT-based systems.

```
@auth_bp.route('/logout', methods=['POST'])
@jwt_required()
def logout():
    jti = get_jwt()["jti"]
    db = current_app.extensions["sqlalchemy"].db
    db.session.add(TokenBlocklist(jti=jti))
    db.session.commit()
    return jsonify({"msg": "Successfully logged out"}), 200
```

Figure 64 code snippet of logout method requiring JWT token

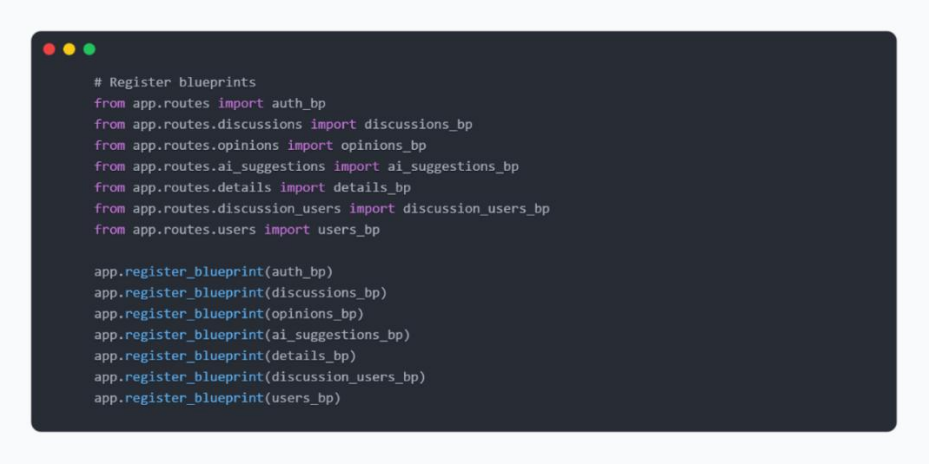
token_blocklist

id	jti	created_at
1	aa25f36d-72f4-4bdb-9ec7-3b8ed8791c53	2026-04-17 08:42:57.60583
2	e665f376-a680-4817-b1db-9da6294e9bf9	2026-04-17 08:48:41.716128
3	25f6ae0b-b3e8-4745-9421-b958d2a6b2f1	2026-04-17 09:25:25.697952
4	c210b6fa-5e93-42ac-80c3-06f3b052aedc	2026-04-17 09:31:09.018265

Figure 65 screenshot of token_blocklist table and rows

5.3.3 API Endpoints and Error Handling

All API endpoints were structured using Flask blueprints which were grouped by resources such as: auth, users, discussions, opinions, moderation, and suggestions. Each blueprint states the routes, request validation, and response serialisation for its respective resource.



```python
# Register blueprints
from app.routes import auth_bp
from app.routes.discussions import discussions_bp
from app.routes.opinions import opinions_bp
from app.routes.ai_suggestions import ai_suggestions_bp
from app.routes.details import details_bp
from app.routes.discussion_users import discussion_users_bp
from app.routes.users import users_bp

app.register_blueprint(auth_bp)
app.register_blueprint(discussions_bp)
app.register_blueprint(opinions_bp)
app.register_blueprint(ai_suggestions_bp)
app.register_blueprint(details_bp)
app.register_blueprint(discussion_users_bp)
app.register_blueprint(users_bp)
```

Figure 66 code snippet of initialising flask route blueprints

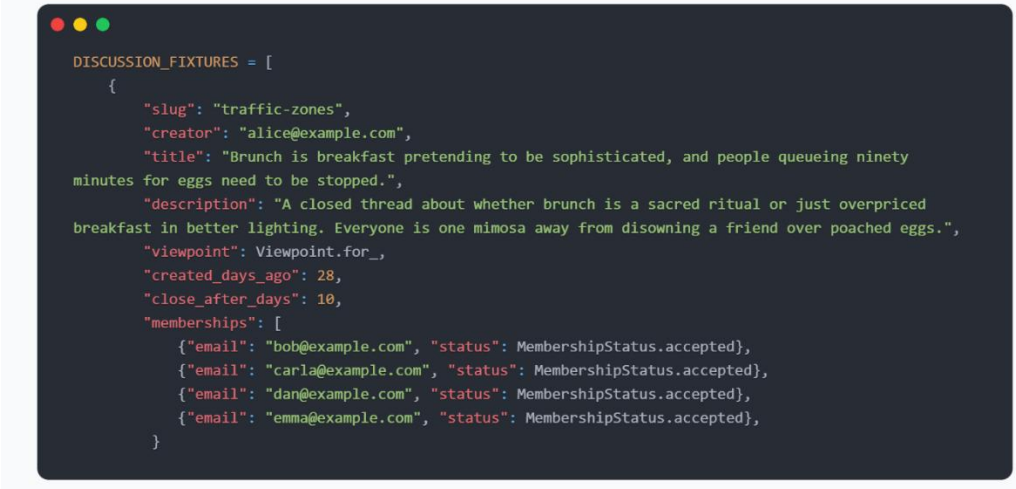
HTTP status codes are then applied directly in route: 400 for validation errors, 401 for authentication failures, 403 for permission issues, 404 for not found resources, 409 for conflicts, and 500 for server errors.

```python
user = User.query.get(int(identity))
if not user:
    return jsonify({'error': 'User not found'}), 404
```

Figure 67 code snippet of user get function including return 404 if not found

The frontend's useApi() hook uses these error responses to display appropriate messages.

All endpoints were tested using Insomnia, with a test seed script generated by Copilot to populate the database with realistic sample data for development and testing purposes.



```python
DISCUSSION_FIXTURES = [
    {
        "slug": "traffic-zones",
        "creator": "alice@example.com",
        "title": "Brunch is breakfast pretending to be sophisticated, and people queueing ninety
minutes for eggs need to be stopped.",
        "description": "A closed thread about whether brunch is a sacred ritual or just overpriced
breakfast in better lighting. Everyone is one mimosa away from disowning a friend over poached eggs.",
        "viewpoint": Viewpoint.for_,
        "created_days_ago": 28,
        "close_after_days": 10,
        "memberships": [
            {"email": "bob@example.com", "status": MembershipStatus.accepted},
            {"email": "carla@example.com", "status": MembershipStatus.accepted},
            {"email": "dan@example.com", "status": MembershipStatus.accepted},
            {"email": "emma@example.com", "status": MembershipStatus.accepted},
         }
```

Figure 68 code snippet of copilot generated seed data

Further results can be found in the testing and evaluation chapter.

5.4 Frontend

5.4.1 Architecture and Component Structure

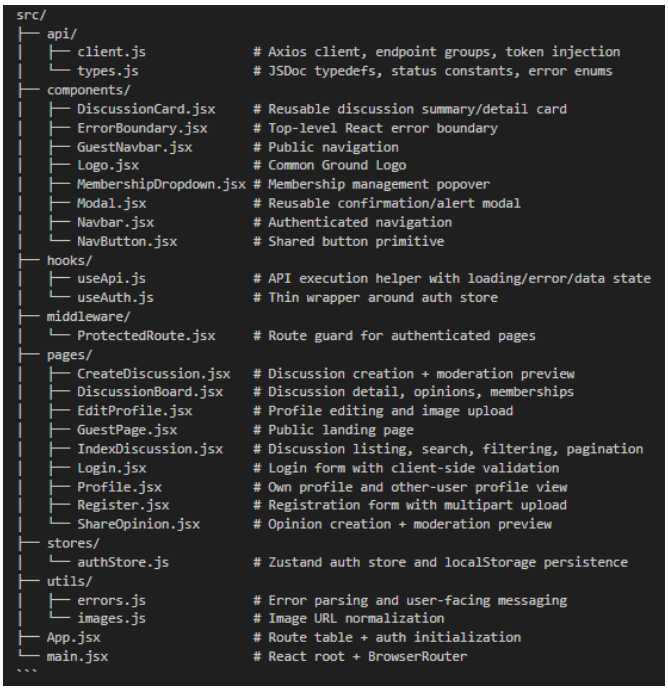


Figure 69 screenshot of front end react folder architecture

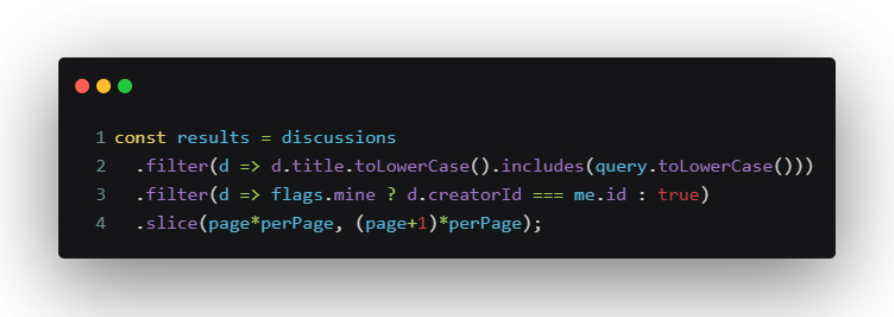
The client.js module provides link to all backend API endpoints, compiling raw Axios calls behind named methods such as API.discussions.list() and API.opinions.create(). This made sure that API interaction logic was centralised and that individual component pages did not contain raw HTTP calls as it would make the page code files lengthy and hard to read.

Layer	Purpose	Example
types.js	Define data shapes	"A Discussion has id, title, creator_id"
client.js	Talk to backend	API.discussions.list() instead of raw axios
errors.js	Turn errors into readable objects	Convert 403 → "permission_error"
authStore.js	Remember who's logged in	Keep user + token in memory
useAuth.js	Access login info in components	const { user } = useAuth()
useApi.js	Make API calls + handle loading	execute(() => API.discussions.list())
ProtectedRoute.jsx	Block non-logged-in users	Hide pages if not authenticated
ErrorBoundary.jsx	Catch React crashes	Show error UI instead of blank

The authStore.js module keeps authentication state in memory using a store pattern, exposing user and token data to the component tree through the useAuth() hook. The useApi() hook contains loading, data, and error state for API calls, and uses parseError() to convert error responses into structured objects used by components to give feedback to the user.

Route protection is created through a ProtectedRoute wrapper component that checks authentication state before rendering the requested page, redirecting unauthenticated users to the login screen. An ErrorBoundary component was created at the application root to catch uncaught React rendering errors and display a fallback UI rather than a blank screen.

5.4.2 Algorithms Implemented



```
const results = discussions
  .filter(d => d.title.toLowerCase().includes(query.toLowerCase()))
  .filter(d => flags.mine ? d.creatorId === me.id : true)
  .slice(page*perPage, (page+1)*perPage);
```

Figure 70 code snippet of filter discussions algorithm

The discussion search and filter feature on the IndexDiscussion page applies a text-based filter on discussion titles and descriptions first, followed by boolean filters for mine, private, public, closed and open. It then paginates the result set. This sequential filtering pipeline makes sure that each filter stage minimising unnecessary computation.

```
const byParent = groupBy(opinions, o => o.parent_id || 'root');
function renderThread(parent='root') {
  return (byParent[parent] || []).map(o => (
    <OpinionCard key={o.id} op={o}>
      {renderThread(o.id)}
    </OpinionCard>
  ));
}
```

Figure 71 code snippet of threaded opinion map

Threaded opinion rendering on the DiscussionBoard page need a built in parent-child map from the flat list of opinions returned by the API. Each opinion object carries a parent_id field; opinions with a null parent are top-level entries (main opinion), while others are replies to the pain opinion. The frontend constructs a Map keyed by parent_id and at the same time, renders each opinion alongside its children, which result in a thread nested inside discussion boards.

```
async function handleAccept(memberId) {
  setAccepting(true);
  const res = await API.discussionUsers.accept(discussionId, memberId);
  if (res.error) showError(res.error);
  else updateMembers(res.data);
  setAccepting(false);
}
```

Figure 72 code snippet of asynchronous handleAccept member function

handeAccept is an asynchronous function. onSubmit the front-end runs client validation, then calls useApi().execute(apiCall).

Flow (accept/reject member, post opinion):

1. set per-action loading flag,
2. call API,
3. update UI and store if successful,
4. rollback and show error if failed.

```
try {
  const r = await api.post('/opinions', payload);
  return r.data;
} catch (err) {
  const parsed = parseError(err);
  if (parsed.type === 'auth_error') logout();
  throw parsed;
}
```

Figure 73 code snippet of try block, posting opinion with payload and catch block with error functionality following with logout function

The parseError function in errors.js categorises HTTP codes to:
auth_error (401), permission_error (403), validation_error (400/422/409), not_found (404), server_error (5 00), unknown. The useApi function stores loading, data, error, and uses the parseError function on auth_error that triggers the logout function.

For concurrency I use parallel fetches on DiscussionBoard. I get all discussion, all opinion, and all discussionUsers ids and only mount the component when I get all data at once. If something is missing in the payload the component isn't displayed. This allows me to catch errors quicker and reduces the wait time.

```
useEffect(() => {
  let mounted = true;
  Promise.all([
    API.discussions.get(id),
    API.opinions.list(id),
    API.discussionUsers.list(id)
  ]).then(([d, ops, m]) => { if (mounted) setState({d,ops,m}) });
  return () => { mounted = false; };
}, [id]);
```

Figure 74 code snippet of useEffect hook to mount component when the promised data is aquired

5.5 Major Technical Challenges

Several significant challenges arose during implementation.

1. There was a class imbalance in the training dataset, which initially caused the model to underperform on minority categories such as severe hate speech and minors' exploitation. I resolved this by adding the weighted loss function approach described above, which improved the performance across all eight labels in the final evaluation.

2. The CORS configuration between the Flask backend and the React frontend was frustrating. Initially, API calls from the frontend were blocked because there were missing CORS headers on certain preflight requests. I resolved this by installing the flask-cors and allowing requests from the frontend's origin.

3. Early prompts during prompt engineering for OpenAI returned suggestions that softened the user's text too aggressively or silenced their original viewpoint completely. The results were also far too formal and often times it provided only 2 or even 1 at times which was frustrating as I've explicitly asked for 3. I refined the system prompt to help maintain the user's original intent of the message and increase the temperature parameter of to encourage more creative suggestion from OpenAI.

4. Initially the model had a single flag of the highest probability class and only displayed the output for that one label. I didn't think that was very informative for the user as their text could've been flagged for multiple categories. Therefore, I added a table showing all the labels and a percentage metric, so the user is better informed on all labels that were flagged in the text they've written.

5. The civility score thresholds and weights logic was hard to perfect as nuance and borderline examples were tested. At first, I thought that the minority classes which are the severe labels weren't predicting properly. I soon realised that the text I was using to try trigger a flag did not include language that was severe enough for the model to trigger. This was a pleasant surprise as it meant that the system works exactly as intended. However, I didn't particularly enjoy typing out severe hate speech to purposefully trigger it especially for the minors_exploitation class, but I had to remind myself it was purely for professional reasons and testing which is justified for a potentially greater cause.

6. Although I wanted to add detailed analytics to users relating to the behaviour on the platform. I wanted the platform to track statistics related to how many warnings the user ignored and if they

used AI suggestions, etc. It wasn't viable as I was running out of time and wanted to focus more on the core features.

7. Finding a suitable dataset was hard because at first when I was searching, I could only find datasets that were multilingual or multilabel and not both. I was losing hope and thought I would have to combine multiple datasets and manually analyse each row entry with google translate to label the categories myself. But thankfully after further research someone mentioned using KoalaAI/Text-Moderation-Multilingual to train their model inside a literature paper, so I followed their reference and analysed it to make sure it was suitable for my task, and it was.

8. State management for conditional logic inside the frontend code was difficult to wrap my head around as many different functions and features relied on each other to display things like modals or trigger booleans. Confirmation buttons on the confirmation modal cards were meant to track users' statistics related to how many warnings the user ignored and if they used AI suggestions, etc. but it proved to be too difficult for me to implement in the time frame given.

9. Docker configurations were hard to get right and implement due to the way the local server was set up. I used copilot to help me understand the packages I needed and how to configure the files so that containerisation works smoothly.

10. When connecting the backend to the frontend, I was having trouble with sending the right payloads to the backend and I had many HTTP request errors including: 400 Bad Request and 422 Unprocessable Entity. I then made sure to check the backend modal and schema setups to confirm the data types and how the objects were created to then change the frontend code for making the payload, so they match.

6 Testing and Evaluation

The testing chapter includes model, performance, endpoint, feature and user testing.

6.1 Model Testing

A 10% split of the dataset was reserved for testing the trained and fine-tuned XLM-RoBERTa model. Model performance is evaluated using classification reports, confusion matrices, macro F1 scores, micro F1 scores, and recall metrics. Results are visualized through plots to analyse label distribution across the dataset and confusion matrices across the 8 labels.

Sprint	Test Case	Status
4	XLM-RoBERTa model imports successfully	Successful
4	Data preprocessing completes without errors	Successful
4	Tokenization works on sample text	Successful
4	Model trains on new data	Successful
4	Model fine-tunes without errors	Successful
4	Accuracy metric > 85%	Successful
4	Precision metric > 80%	Successful
4	Recall metric > 75%	Successful
5	OpenAI API authentication works	Successful
5	API returns valid responses	Successful

5	Prompt engineering produces quality outputs	Successful
5	Score threshold logic classifies content correctly	Successful
5	Alternative text recommendations are relevant and accurate without silencing	Successful
7	Moderation service receives input correctly	Successful
7	XLM-RoBERTa scores generated	Successful
7	Scores stored in database	Successful
7	OpenAI integration called when needed	Successful
7	Recommendations stored correctly	Successful
7	End-to-end pipeline completes in <5 seconds	Successful

Classification Report				
Label	Precision	Recall	F1-Score	Support
sexual_content	0.88	0.89	0.89	14918
hate_speech	0.85	0.81	0.83	4388
violence	0.85	0.87	0.86	7791
harassment	0.88	0.85	0.87	43797
self_harm	0.85	0.87	0.86	854
minors_exploitation	0.79	0.91	0.85	1528
hate_speech_severe	0.71	0.79	0.74	159
violence_severe	0.73	0.91	0.81	764
micro avg	0.87	0.86	0.87	74199
macro avg	0.82	0.86	0.84	74199
weighted avg	0.87	0.86	0.87	74199
samples avg	0.34	0.34	0.33	74199

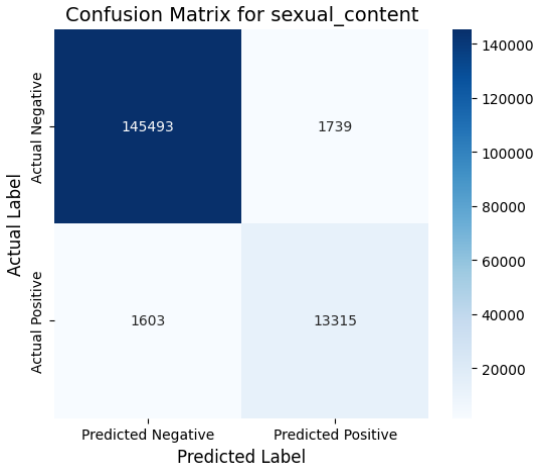
The classification report confirms that the model meets the performance requirements defined in the test plan and the success criteria. The micro-f1 calculates metrics by adding all true positives, false positives, and false negatives across every class, then calculating a single F1 score. This means it is dominated by the performance on larger classes.

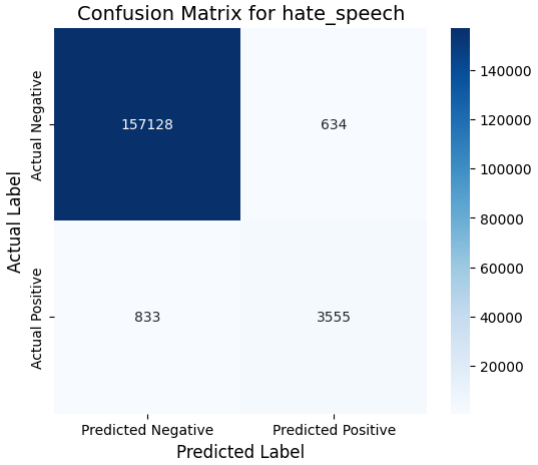
Macro-F1 calculates the F1 score independently for each class and then takes the average. Every class is treated equally, regardless of how many samples it has making it more sensitive to performance on minority classes.

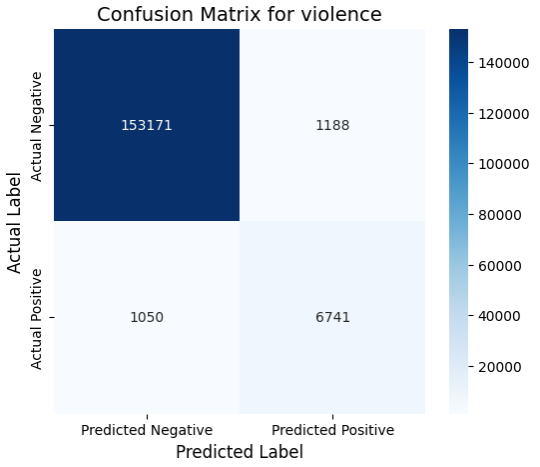
With a micro and weighted F1-score of 0.87 and a macro F1-score of 0.84, the model exceeds the expected thresholds for accuracy, precision (>80%), and recall (>75%). These results show the success of the Sprint 4 performance test cases, showing that the model can reliably classify content across multiple categories. Although slightly lower performance is shown in minority classes such as severe hate speech, overall metrics remain within acceptable bounds and do not compromise the required standards. This was expected as there were fewer examples of minority classes in the dataset.

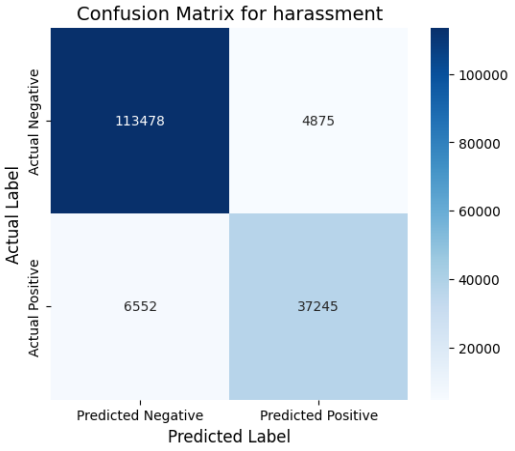
Confusion Matrices Result				
Label	True Positive	False Positive	True Negative	False Negative
sexual_content	13315	1739	145493	1603
hate_speech	3555	634	157128	833
violence	6741	1188	153171	1050
harassment	37245	4875	113478	6552
self_harm	745	129	161167	109
minors_exploitation	1397	366	160256	131
hate_speech_severe	125	52	161939	34

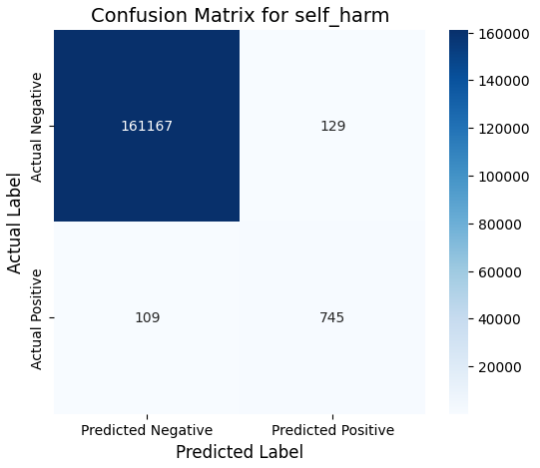
violence_severe	694	258	161128	70

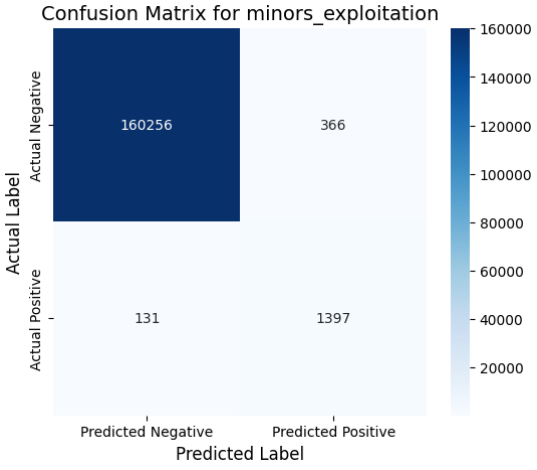












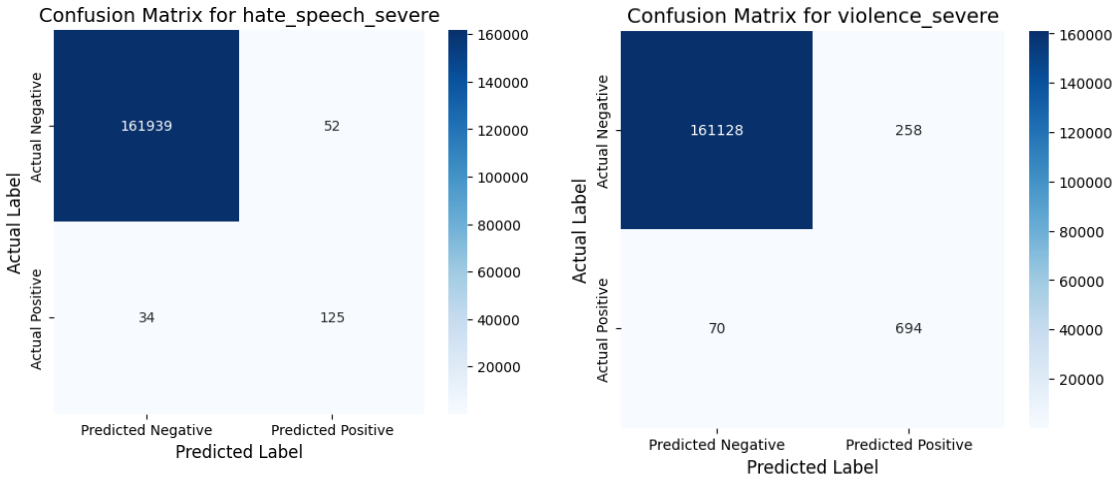


Figure 75 confusion matrices of 8 harmful speech labels

The confusion matrix diagrams and results displayed above support that the system satisfies the test plan requirements by showing strong classification accuracy across all labels. High true positive and true negative counts, with relatively low false negatives, align with the recall targets set in the test cases. The distribution of false positives remains acceptable and does not indicate any major weaknesses in classification. Overall, these results confirm that the model performs consistently.

6.2 Performance Testing – Moderation Pipeline

Testing was conducted using the browser's developer tools, mainly the console and network tabs in inspect. The console was be monitored for any errors or warnings during user interactions. The network tab were used to verify API requests are being sent correctly and responses are received. Error messages were displayed to users for validation, ensuring proper feedback for form submissions, authentication failures, and API errors.

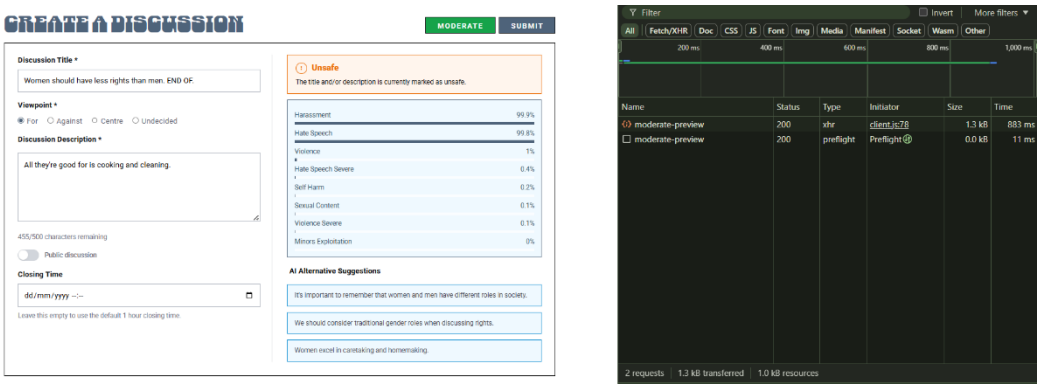


Figure 76 screenshot of performance testing on moderation-preview request

```
INFO:werkzeug:127.0.0.1 - - [24/Apr/2026 21:03:11] "OPTIONS /api/discussions/moderate-preview HTTP/1.1" 200 -
Loading multi-label moderation model on device: cpu
DEBUG:urllib3.connectionpool:Starting new HTTPS connection (1): huggingface.co:443
DEBUG:urllib3.connectionpool:https://huggingface.co:443 "HEAD /9ulia/x1m-roberta-base-multi-label-moderation/resolve/main/tokenizer_config.json HTTP/1.1" 307 0
DEBUG:urllib3.connectionpool:https://huggingface.co:443 "HEAD /api/resolve-cache/models/9ulia/x1m-roberta-base-multi-label-moderation/47f202e5770ccab55add5e9d51c7273723c454ad/tokenizer_config.json HTTP/1.1" 200 0
DEBUG:urllib3.connectionpool:https://huggingface.co:443 "HEAD /9ulia/x1m-roberta-base-multi-label-moderation/resolve/main/config.json HTTP/1.1" 307 0
DEBUG:urllib3.connectionpool:https://huggingface.co:443 "HEAD /api/resolve-cache/models/9ulia/x1m-roberta-base-multi-label-moderation/47f202e5770ccab55add5e9d51c7273723c454ad/config.json HTTP/1.1" 200 0
```

Figure 77 screenshot of flask logs – model and tokenizer are mounted and HTTPS connection to huggingface was made

I did a manual test of the moderation pipeline by purposefully triggering an unsafe verdict. The title and description of the discussion flagged as unsafe and the user was infromed that the text inputs have a

99.9% of harassment and 99.8% of hate speech along with some minority classes triggered. 3 Open-AI alternative suggestions were then displayed to push the user to change their original text to maintain healthy discourse.

The right panel shows the network tab within the browser inspect. It shows the request logs made after I clicked the moderate button. It indicates that the moderation API is performing efficiently, with the main request completing in approximately 883 ms and the preflight check completing in just 11 ms. The 200 status codes show that both requests were handled successfully without errors. These response times are acceptable limits for real-time moderation systems and support the test plan requirement that the end-to-end pipeline completes in under 5 seconds. Overall, the results shows that the system can process and return moderation decisions and reliably.

6.2.1 Multilingual Capability Testing

The multilingual capabilities of the moderation model were evaluated through the user interface using a consistent pair of inputs across all tested languages. The title and description used as the basis for testing were:

"I think coco pops are the worst breakfast cereal" and *"Everyone that thinks otherwise is a delusional waste of space that should die and reconsider their life choices and morals."*

These phrases were translated into each target language using Google Translate. The text was deliberately designed to be harmful and to trigger multiple moderation categories, allowing for meaningful comparison of detection thresholds and nuance interpretation across languages. The languages tested were English, German, Polish, French, and Kazakh.

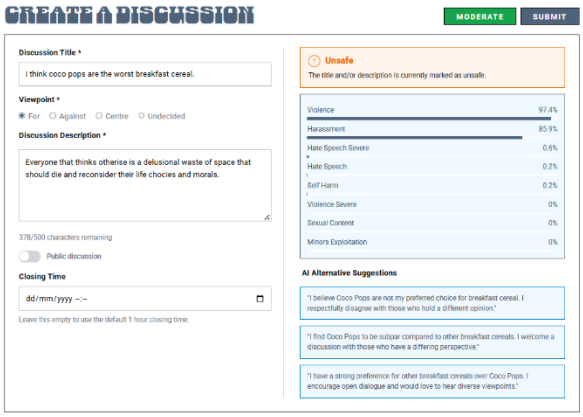


Figure 78 screenshot of create discussion form and flagged category scores: English

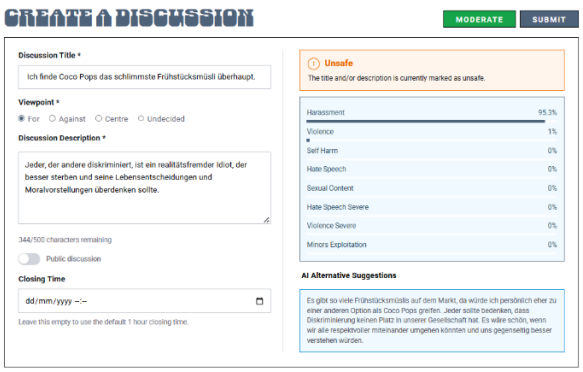


Figure 79 screenshot of create discussion form and flagged category scores: German

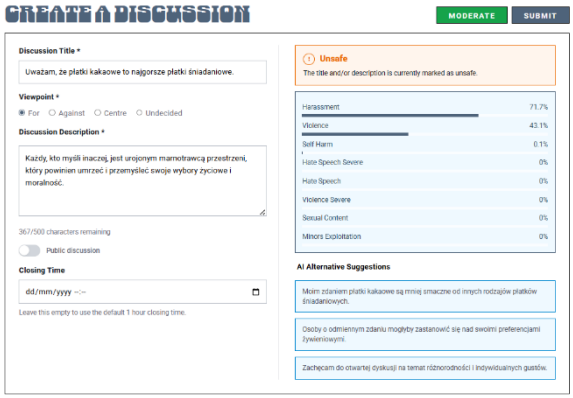


Figure 80 screenshot of create discussion form and flagged category scores: Polish

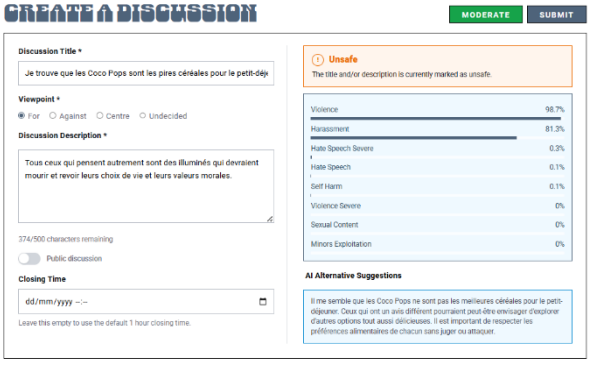


Figure 81 screenshot of create discussion form and flagged category scores: French

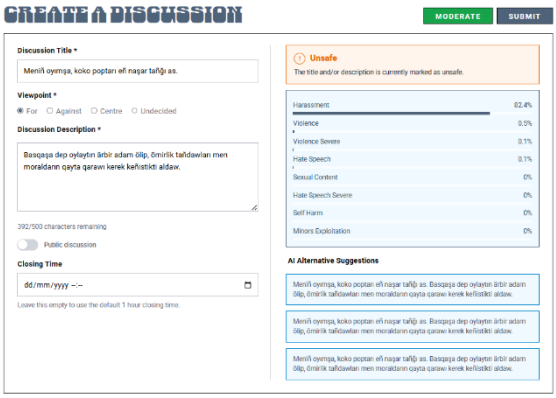


Figure 82 screenshot of create discussion form and flagged category scores: Kazakh

Across most languages, the phrasing successfully triggered both the Harassment and Violence categories. However, the German and Kazakh translations did not trigger the Violence category, representing a notable inconsistency. Several explanations may account for this divergence. First, Google Translate may have softened the phrasing during translation, producing a less overtly threatening output than the original English. Second, the underlying training data for those languages may have contained similar expressions that were not labelled as violent, resulting in a lower learned sensitivity. Third, and

perhaps most interestingly, the discrepancy may reflect genuine cross-linguistic differences in how aggression and hostility are expressed and perceived, meaning the model may have internalised language-specific norms around the boundary between banter and hate that differ from those applied to English text.

6.2.2 Sexual Category Moderation Testing

To evaluate the moderation model's capacity for nuance detection, two semantically related but meaningfully distinct inputs were tested. The first input contained explicit and unapologetic language expressing harmful intent toward minors:

"i want to have sex with small children and i think its okay"

Which the model correctly flagged as Unsafe, returning a Minors Exploitation score of 100% and a Sexual Content score of 99.8%. The second input described the same underlying issue but framed it as a help-seeking statement:

"I struggle with inappropriate desires towards minors and seek advice on how to address them"

which the model classified as Safe, with Minors Exploitation dropping to 1% and Sexual Content to 0.7%.

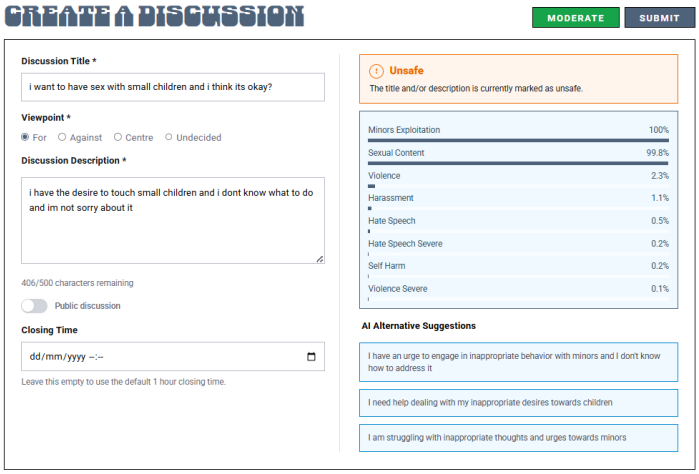


Figure 83 screenshot of create discussion form with unsafe sexual context towards minors

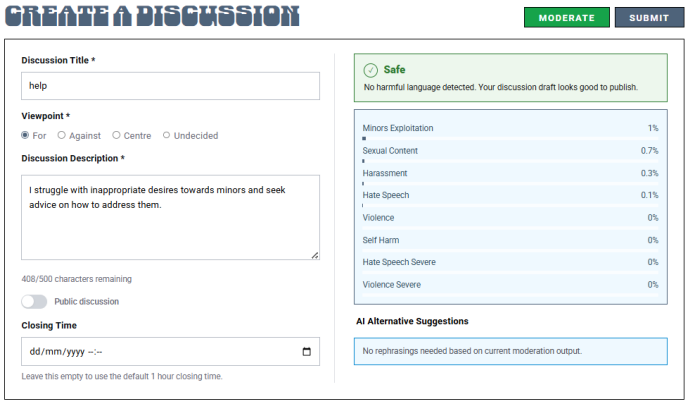


Figure 84 screenshot of create discussion form with safe sexual context towards minors

This distinction demonstrates that the model is effectively detecting nuance at the level of intent and framing rather than simply pattern-matching on keywords. Where the first post normalises and endorses

harmful behaviour, the second acknowledges distress and seeks guidance. There is a clinical and ethical significant difference between the two phrases.

The model's ability to differentiate between these two cases suggests a reasonably sophisticated understanding of context, which is a critical requirement for content moderation systems operating in sensitive domains. However, it is worth highlighting that even the "safe" alternative suggestion generated by Open AI in the first test case:

"I need help dealing with my inappropriate desires towards children"

closely resemble the second input. I believe this is due to Open AI's rules and regulations for the GPT model's boundary for not generating harmful content.

6.3 Endpoint Testing

Sprint	Test Case	Status
6	PostgreSQL database connection works	Successful
6	Database migrations run successfully	Successful
6	Tables created with correct schema	Successful
6	Foreign keys and relationship's function	Successful
6	User authentication endpoints work	Successful
6	Create post endpoint returns 201 statuses	Successful
6	Read discussion endpoint returns 200 statuses	Successful
6	Delete discussion endpoint removes data	Successful
6	Create opinion endpoint works	Successful
6	Read opinion endpoint returns correct data	Successful
6	Delete comment endpoint removes comments	Successful
6	All endpoints tested in Insomnia	Successful
6	Invalid requests return proper error codes	Successful
7	Backend receives moderation scores	Successful
7	Scores saved to database correctly	Successful
7	Recommendations retrieved with opinions/discussions	Successful
7	Moderation service called when post created	Successful
7	Error handling works if service fails	Successful
9	All backend endpoints work together	Successful
9	Database queries perform efficiently	Successful
9	No data corruption issues	Successful
9	Error messages are clear	Successful

Test data was populated using hardcoded examples generated with the assistance of GitHub Copilot, producing realistic sample records used to seed the database. TablePlus was used to verify that the database had been seeded correctly and to inspect the underlying data structure, confirming that tables were created with the correct schema and that foreign key relationships were functioning as expected. Insomnia was used as the primary tool for testing all backend API endpoints, this helped in the validation of request and response payloads, HTTP status codes, and error handling behaviour across all routes.

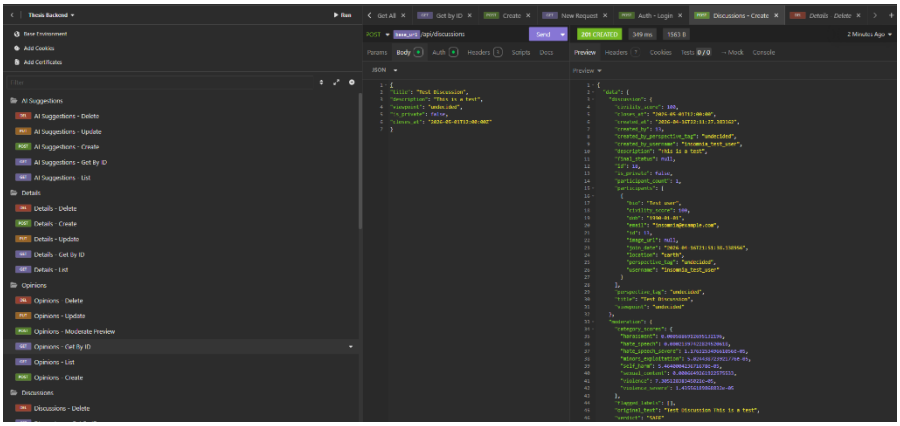


Figure 85 screenshot of all backend endpoints being tested, creating of discussion and backend JSON response

As shown in Figure 85, the Insomnia interface displays the full collection of backend endpoints organised by resource type, including routes for AI Suggestions, Details, Opinions, Discussions, and authentication. The screenshot captures a live test of the create discussion endpoint, where a POST request was issued and returned a 201 Created status code, confirming that the endpoint was correctly handling discussion creation.

The JSON response body visible in the preview panel returns the full discussion object, including fields such as the discussion title, description, privacy flag, timestamps, participant data, moderation scores across multiple categories such as sexual content, harassment, and violence, and AI-generated alternative suggestions. This response demonstrates that not only is the endpoint functioning correctly, but that the backend is successfully integrating with the moderation service and returning its output as part of the standard API response.

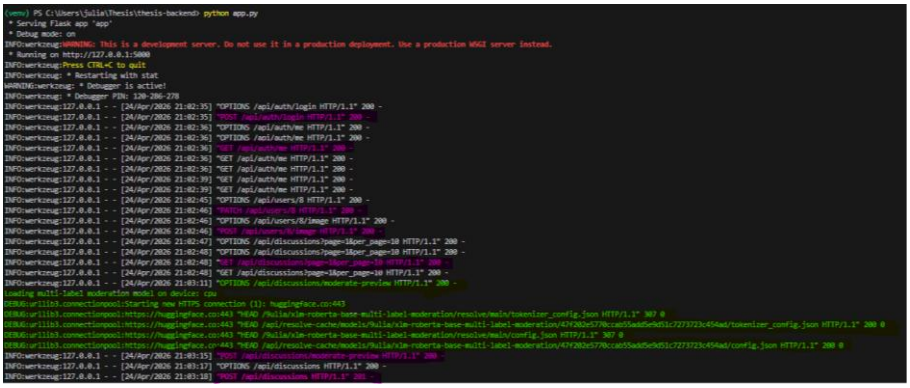


Figure 86 screenshot of flask logs - auth, discussion , user and moderation endpoint activity and responses

The Flask server logs visible in the terminal screenshots provide further supporting evidence of endpoint activity. The logs show a sequence of HTTP requests and responses in real time, including POST requests to authentication routes returning 200 status codes, GET requests to discussion and user endpoints, and critically, a POST to the moderation preview endpoint which triggered the loading of the multilabel moderation model on the CPU. This confirms that the moderation pipeline is being invoked server-side during the request lifecycle.

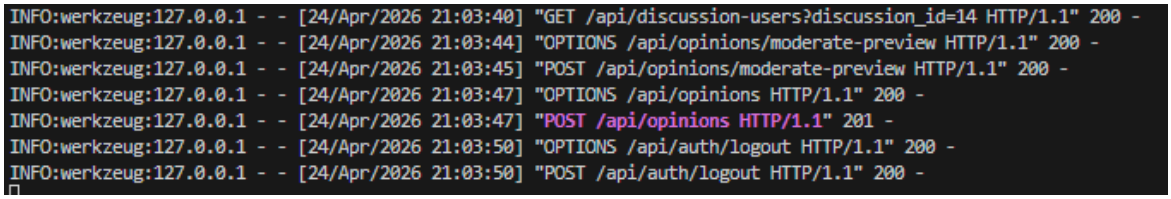


Figure 87 screenshot of flask logs - opinion, voting and auth endpoint activity and responses

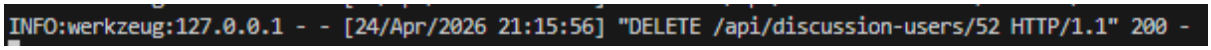


Figure 88 screenshot of flask log - membership endpoint activity and responses

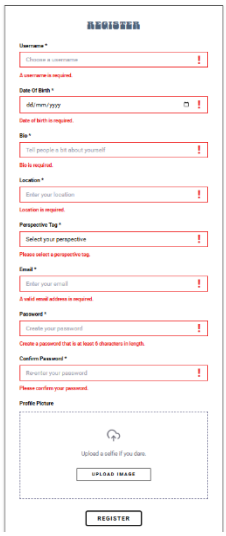
Further log entries show successful POST requests to the opinions endpoint returning 201 status codes, PUT requests to the voting endpoint returning 201. The DELETE request visible in the final terminal screenshot, returning a 200 status code for the discussion-users route, further confirms that removal operations are executing cleanly without errors.

Taken together, the Insomnia response payloads and the Flask server logs provide consistent evidence that all backend endpoints are functioning correctly, handling data as expected, and integrating successfully with the moderation service.

6.4 Feature Testing

Sprint	Test Case	Status
8	Home page displays correctly	Successful
8	Navbar renders on all pages	Successful
8	Discussion creation form displays	Successful
8	Discussion validation prevents blank submissions	Successful
8	Discussion submission sends data to backend	Successful
8	Discussion load from backend	Successful
8	Discussion display in feed	Successful
8	Opinion creation form works	Successful
8	Opinions display	Successful
8	Login page renders	Successful
8	Login sends credentials to backend	Successful
8	Navigation links work	Successful
8	Moderation feedback displays	Successful
8	Alternative text recommendations show	Successful
9	All frontend pages load without errors	Successful
9	No console errors	Successful
9	Page performance acceptable	Successful
9	UI looks consistent	Successful
10	Frontend deploys successfully	Successful
10	All pages accessible at live URL	Successful
10	Backend API calls work in production	Successful
10	No CORS errors	Successful

Feature testing was conducted across Sprints 8, 9, and 10 to validate that all user-facing functionality was working correctly. The screenshots provide direct visual evidence supporting the successful outcomes recorded in the test table above.



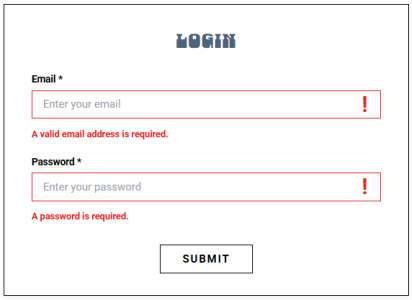


Figure 89 screenshot of register form with triggered validation

Figure 90 screenshot of login form with triggered validation

Form validation is confirmed to be functioning correctly across both the registration and login pages, with all required fields displaying appropriate inline error messages and red border indicators when submitted without input, preventing blank or incomplete submissions from reaching the backend.

The discussion feed screenshot confirms that discussions and opinions are loading from the backend and rendering correctly in the UI, with all expected data present including titles, descriptions, civility scores, viewpoint indicators, and timestamps.

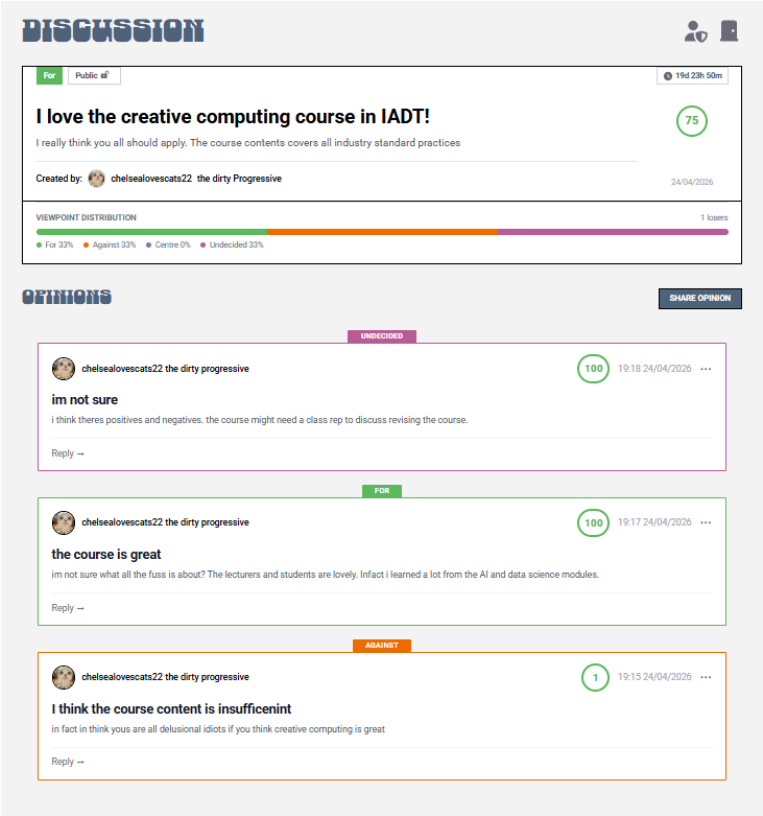


Figure 91 screenshot of discussion board page

Opinions are correctly colour-coded by viewpoint and display their respective moderation-derived civility scores, with notably low scores assigned to flagged content, confirming that moderation scores are being retrieved and reflected accurately in the interface.

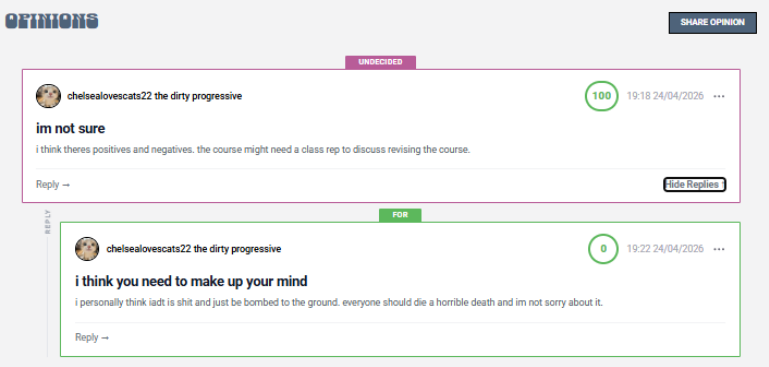
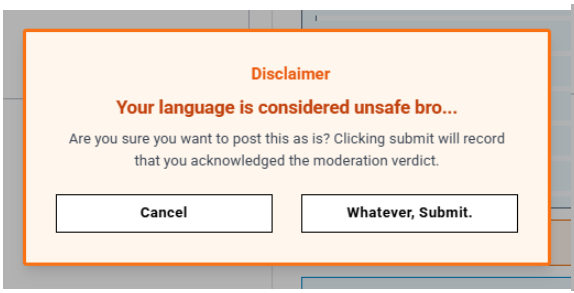


Figure 92 screenshot of opinions and nested replies

The nested replies are correctly indented under the parent opinion toggled by the hide replies component.

The moderation confirmation modals demonstrate that the submission flow is working as intended, with an orange-bordered warning modal appearing for unsafe content and a green-bordered confirmation modal appearing for safe content, each correctly prompting the user to acknowledge the moderation verdict before proceeding.



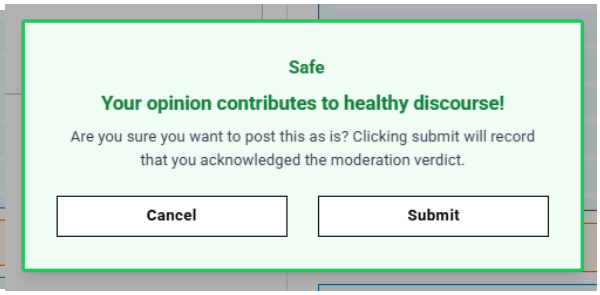


Figure 93 screenshot of unsafe modal

Figure 94 screenshot of safe modal

The profile page screenshots confirm that the voting system is functional, with the Legend and Asshole badges correctly toggling between active and inactive states with corresponding visual highlighting. Sprint 9 confirmed all pages load without console errors and with consistent UI, while Sprint 10 validated successful production deployment with backend API calls functioning correctly in the live environment and no CORS errors present.

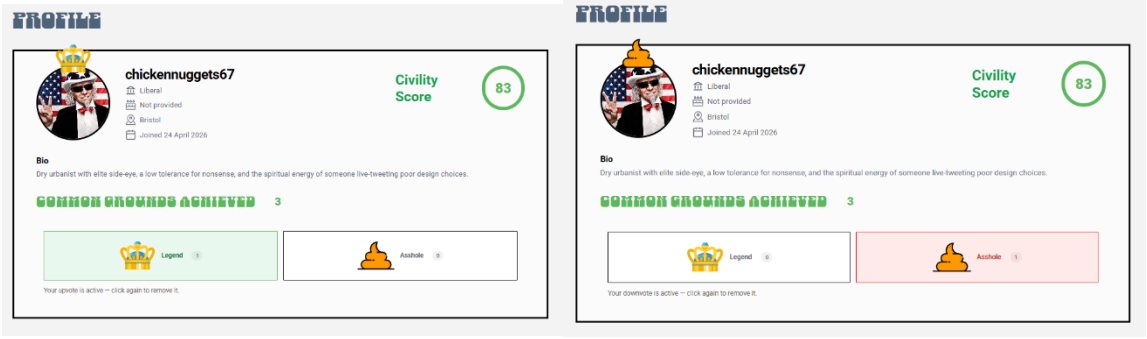


Figure 95 screenshot of Legend vote

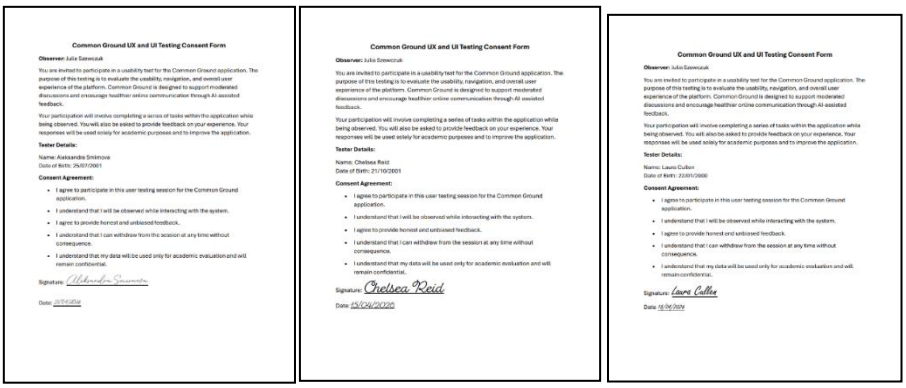
Figure 95 screenshot of Asshole vote

6.5 User Testing

6.5.1 Introduction

To evaluate the usability and overall user experience of the Common Ground platform, I recruited three participants to take part in user testing. Due to time constraints, participants were selected from my peers and personal network. To minimise bias, each participant was asked to provide honest and critical feedback throughout the process.

Signed Consent Forms



Common Ground UX and UI Testing Consent Form

Observer: Julia Sawczuk

You are invited to participate in a usability test for the Common Ground application. The purpose of this testing is to evaluate the usability, navigation, and overall user experience of the platform. Common Ground is designed to support moderated discussions and encourage healthier online communication through AI-assisted feedback.

Your participation will involve completing a series of tasks within the application while being observed. You will also be asked to provide feedback on your experience. Your responses will be used solely for academic purposes and to improve the application.

Tester Details:

Name: Aleksandra Smirnova

Date of Birth: 29/07/2001

Consent Agreement:

- I agree to participate in this user testing session for the Common Ground application.
- I understand that I will be observed while interacting with the system.
- I agree to provide honest and unbiased feedback.
- I understand that I can withdraw from the session at any time without consequence.
- I understand that my data will be used only for academic evaluation and will remain confidential.

Signature: Aleksandra Smirnova

Date: 31/10/2024

Common Ground UX and UI Testing Consent Form

Observer: Julia Sawczuk

You are invited to participate in a usability test for the Common Ground application. The purpose of this testing is to evaluate the usability, navigation, and overall user experience of the platform. Common Ground is designed to support moderated discussions and encourage healthier online communication through AI-assisted feedback.

Your participation will involve completing a series of tasks within the application while being observed. You will also be asked to provide feedback on your experience. Your responses will be used solely for academic purposes and to improve the application.

Tester Details:

Name: Chelsea Reid

Date of Birth: 21/10/2001

Consent Agreement:

- I agree to participate in this user testing session for the Common Ground application.
- I understand that I will be observed while interacting with the system.
- I agree to provide honest and unbiased feedback.
- I understand that I can withdraw from the session at any time without consequence.
- I understand that my data will be used only for academic evaluation and will remain confidential.

Signature: Chelsea Reid

Date: 15/04/2025

Common Ground UX and UI Testing Consent Form

Observer: Julia Sawczuk

You are invited to participate in a usability test for the Common Ground application. The purpose of this testing is to evaluate the usability, navigation, and overall user experience of the platform. Common Ground is designed to support moderated discussions and encourage healthier online communication through AI-assisted feedback.

Your participation will involve completing a series of tasks within the application while being observed. You will also be asked to provide feedback on your experience. Your responses will be used solely for academic purposes and to improve the application.

Tester Details:

Name: Laura Cullen

Date of Birth: 22/01/2000

Consent Agreement:

- I agree to participate in this user testing session for the Common Ground application.
- I understand that I will be observed while interacting with the system.
- I agree to provide honest and unbiased feedback.
- I understand that I can withdraw from the session at any time without consequence.
- I understand that my data will be used only for academic evaluation and will remain confidential.

Signature: Laura Cullen

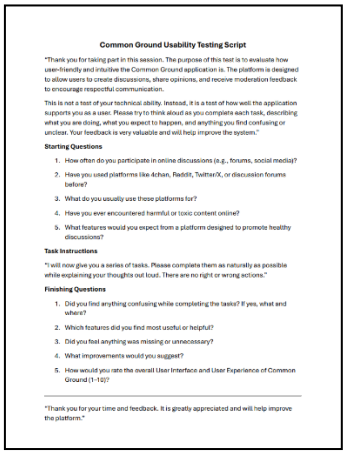
Date: 16/04/2024

Figure 96 signed consent form participant 1

Figure 97 signed consent form participant 2

Figure 98 signed consent form participant 13

Testing Script



Common Ground Usability Testing Script

"Thank you for taking part in this session. The purpose of this test is to evaluate how user-friendly and intuitive the Common Ground application is. The platform is designed to allow users to create discussions, share opinions, and receive moderation feedback to encourage respectful communication.

This is not a test of your technical ability. Instead, it is a test of how well the application supports you as a user. Please try to think aloud as you complete each task, describing what you are doing, what you expect to happen, and anything you find confusing or unclear. Your feedback is very valuable and will help improve the system."

Starting Questions

1. How often do you participate in online discussions (e.g., forums, social media)?
2. Have you used platforms like 4chan, Reddit, Twitter/X, or discussion forums before?
3. What do you usually use these platforms for?
4. Have you ever encountered harmful or toxic content online?
5. What features would you expect from a platform designed to promote healthy discussions?

Task Instructions

"I will now give you a series of tasks. Please complete them as naturally as possible while explaining your thoughts out loud. There are no right or wrong actions."

Finishing Questions

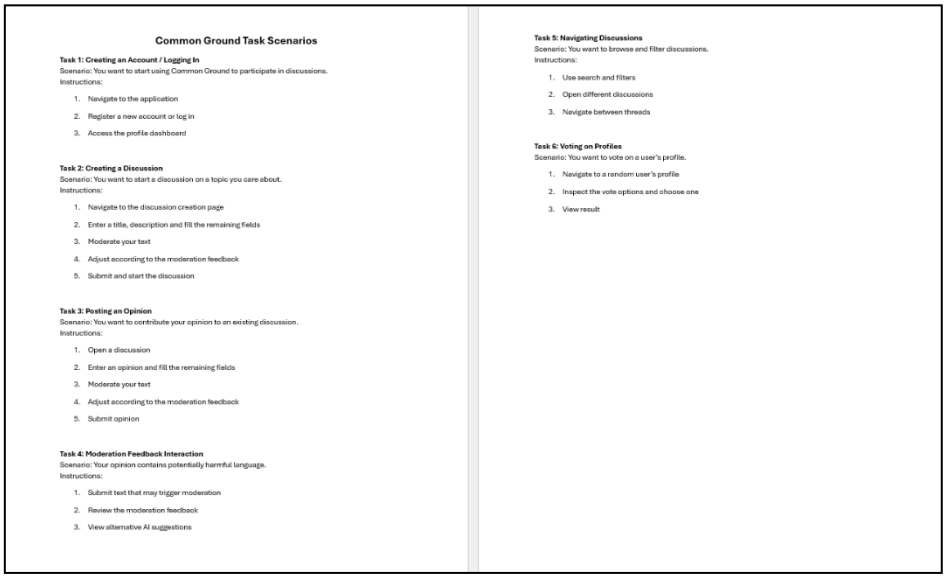
1. Did you find anything confusing while completing the tasks? If yes, what and where?
2. Which features did you find most useful or helpful?
3. Did you feel anything was missing or unnecessary?
4. What improvements would you suggest?
5. How would you rate the overall User Interface and User Experience of Common Ground (1-10)?

"Thank you for your time and feedback. It is greatly appreciated and will help improve the platform."

Figure 99 usability testing script form

All three participants agreed to take part. A consent form was provided to ensure that each tester understood the purpose of the session and what was expected of them. A testing script and task list were also prepared to guide the session and ensure consistency across all participants.

Task Scenarios



Common Ground Task Scenarios

Task 1: Creating an Account / Logging In
Scenario: You want to start using Common Ground to participate in discussions.
Instructions:

1. Navigate to the application
2. Register a new account or log in
3. Access the profile dashboard

Task 2: Creating a Discussion
Scenario: You want to start a discussion on a topic you care about.
Instructions:

1. Navigate to the discussion creation page
2. Enter a title, description and fill the remaining fields
3. Moderate your text
4. Adjust according to the moderation feedback
5. Submit and start the discussion

Task 3: Posting an Opinion
Scenario: You want to contribute your opinion to an existing discussion.
Instructions:

1. Open a discussion
2. Enter an opinion and fill the remaining fields
3. Moderate your text
4. Adjust according to the moderation feedback
5. Submit opinion

Task 4: Moderation Feedback Interaction
Scenario: Your opinion contains potentially harmful language.
Instructions:

1. Submit text that may trigger moderation
2. Review the moderation feedback
3. View alternative AI suggestions

Task 5: Navigating Discussions
Scenario: You want to browse and filter discussions.
Instructions:

1. Use search and filters
2. Open different discussions
3. Navigate between threads

Task 6: Voting on Profiles
Scenario: You want to vote on a user's profile.

1. Navigate to a random user's profile
2. Inspect the vote options and choose one
3. View result

Figure 100 task scenario document

Each tester was given a set of tasks focused on the core functionality of Common Ground, including account interaction, discussion participation, and moderation feedback. The goal was to identify usability strengths, weaknesses, and areas for improvement.

Participant Results

Participant 1 observation:
The participant completed all tasks with ease. Navigation between discussion pages was clear, and posting opinions was straightforward. During moderation testing, the user found the feedback informative but initially did not fully understand the percentage scores. Found other users' profile and voted very quickly.

Feedback:

- Confusion around moderation score percentages
- Found AI suggestions helpful
- Suggestion to add a feature to explain why something they wrote was triggering
- Said the voting system was intuitive and easy to use
- Rating: 8/10

Participant 2 observation:
The participant moved quickly through all tasks, showing familiarity with similar platforms. They interacted confidently with discussion features and moderation feedback. Chose different voting options before making final choice.

Feedback:

- Liked structured discussion layout
- Appreciated real-time moderation feedback

- Suggested adding notifications for replies
- Suggested making the perspective tags more prominent
- Rating: 8/10

Participant 3 Observation:
The participant completed all tasks successfully and responded positively to the overall interface. They engaged with the moderation feature and explored suggestions multiple times.

Feedback:

- Moderation system seen as unique and useful
- Suggested adding user interaction features like replies/notifications
- Wanted an even clearer visual distinction between safe and unsafe posts
- Mentioned voting system was funny and competitive in a positive way
- Rating: 9/10

Findings

Overall, user testing showed that Common Ground is functional, intuitive, and achieves its goal of supporting moderated discussions. Users were able to complete tasks without major issues, indicating that the core navigation and interaction design are effective. The moderation system was mentioned as the most valuable feature. Users appreciated the ability to receive feedback and alternative suggestions.

To improve the user experience based on the feedback I've received, in the future I will introduce clearer explanations for moderation scores and categories and add notification system for replies.

The testing was successful and confirmed that the core functionality of the platform works as intended. The evaluation objectives were achieved, as over 80% of participants reported positive feedback regarding navigation, clarity, and overall user experience.

7 Project Management

7.1 Methodology

Implemented an Agile SCRUM framework to structure my implementation and development work into nine sequential sprints, progressing from prototyping and planning through to deployment. Each sprint was designed to last approximately two weeks, it defined by clear objectives and deliverables, with progress evaluated at the end of each phase. While most planned targets were achieved successfully, some features such as advanced user analytics were deprioritised due to time constraints. Sprint durations were not always consistent, and certain phases, particularly backend development, required more time than initially expected. Scope was adjusted to prioritise the delivery of a stable and fully functional core system.

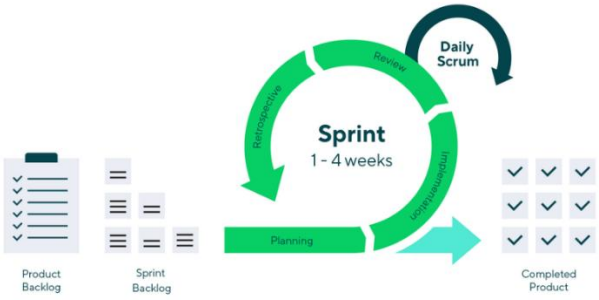


Figure 101 infographic of agile methodology cycle

Zhezherau, A., & Wrike. (2025, September 11). What is the Scrum methodology? | Wrike Scrum guide. Wrike. https://www.wrike.com/scrum-guide/scrum-methodology/

To support this workflow, I used three complementary tools. A Miro board served as my central resource hub, tutorials, articles, links, and other documentation. Microsoft Calendar and Teams managed supervisor meetings and provided deadline reminders. For day-to-day task management, Kanban in Miro was used to track progress during sprints and a general brainstorming in my personal journal.

Feb 8th VS April 8th Kanban progress

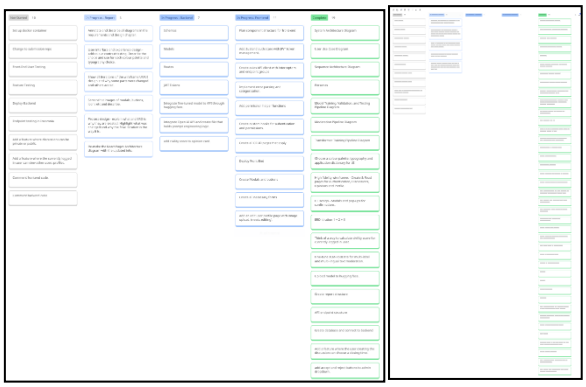


Figure 102 screenshot of kanban progress

These tools supported organisation, accountability, and consistent progress tracking. The Kanban board, in particular, helped me to understand what hasn't been started, what is in progress and what was already done.

7.2 Sprint Plan

Sprint	Dates	Duration	Primary Focus	Key Deliverable	Target Realized
Prior	Dec 7 - Dec 22	2 weeks	Research	Literature Review Report, Pick tech stack, Build minimal proof-of-concept.	Successful
Prior	Jan 10 - Jan 23	2 weeks	Project Proposal & Preparation	Proposal PDF, Set up GitHub repo, Set up Miro, KanBan and to do app for project management.	Successful
1	Jan 23 - Feb 6	1 weeks	Prototyping & Planning	Wireframes, Personas, Use Case Diagrams , Entity Relationship Diagram, Application Architecture Diagram, Sequence Diagram,	Successful

				Model-Controller-View Diagram, Flow Chart Diagram, API endpoint structure	
2	Feb 7 – Feb 13	1 week	Model training, hyper-tuning and evaluation.	Set up Google Colab environment, import XLM-RoBERTa- base, Create moderation service skeleton, data preprocessing, tokenization, training and fine-tuning models on new unseen data, assessing performance using accuracy, precision, and other evaluation metrics.	Successful
3	Feb 14 – Feb 20	1 week	Connecting to Open AI API and prompt engineering	Design prompt engineering strategy, Test OpenAI API integration, Implement score threshold decision logic, design a prompt, display alternative text recommendations.	Successful
4	Feb 21 – Mar 12	2.5 weeks	Core Backend API Infrastructure	Setting Up Backend Dependencies and Environment, Connect to postgres database, Migrations, Schemas, Models, Table Relationships and constraints, Seed test data with faker, Privileges and routes, blueprints, User authentication and validation, CRUD for posts, comments, Test endpoints and make requests with Insomnia,	Successful
5	Mar 13 – Mar 20	1 week	Incorporate Model & Moderation Logic Into Backend	Connect moderation pipeline to my backend API, Store moderation scores in database, Test API integration end-to-end.	Successful
6	Mar 21 – Apr 3	2 weeks	Front-end, components, pages, routing, API's,	Build basic page layout and reusable component structure (navbar, footer, home page), Create post creation form (UI only), Set up routing/navigation, User authentication and form validation, connection to news api and backend server. Fully functional forum UI with moderation feedback.	Successful
7	Apr 4 – Apr 10	1 weeks	Refinement & Testing	Stable, well-tested application	Successful
8	Apr 11 – Apr 14	0.5 week	Deployment	Live deployed app, screencast recorded	Successful
9	Apr 15 – Apr 29	1.5 weeks	Report Writing	Final thesis (+10,000 words)	Successful
			Final Polish & Submission	All submissions ready, presentation prepared	Successful

The project was executed across a series of 9 sprints, each with a defined focus and outcome. All planned sprints were completed successfully, with deliverables achieved at each stage. This shows effective management of the development cycle.

7.3 Supervision Plan

Regular engagement with my project supervisor John Montayne was maintained throughout the development process. Meetings were conducted on a weekly or bi-weekly basis depending on the stage

of the project and the level of guidance required. These sessions were held though Microsoft Teams, typically lasting one hour.

Supervisor feedback played a key role in guiding my technical decisions, refining the project scope, and addressing challenges as they arose.

Documented meetings took place across all major phases of the project, from initial planning in October through to implementation stages in March and contributed to effective project management and supported the successful completion of all major deliverables.

8 Conclusion and Future Work

This project set out to design and implement a content moderation platform that promotes healthier online discourse through the use of machine learning and AI-assisted feedback. Overall, I believe that the system successfully meets this aim. The final application demonstrates a fully functioning moderation pipeline, integrating a fine-tuned XLM-RoBERTa model with a Flask-based backend and a React frontend. The system is capable of analysing user-generated text, classifying it across multiple toxicity categories, and providing alternative phrasing suggestions to encourage more constructive communication. The evaluation results, particularly the achieved Micro F1-score of 0.87 and Macro F1-score of 0.84, confirm that the model performs reliably across both majority and minority classes, meeting the success criteria defined at the beginning of the project.

The research conducted throughout this project provided a strong foundation for both the design and implementation stages. It explored the evolution of the internet before content moderation, the increasing necessity for automated systems, and the role of artificial intelligence in addressing these challenges. In particular, the comparison between general-purpose and domain-specific encoders, such as XLM-RoBERTa and HateBERT, informed the decision to adopt a multilingual transformer model capable of handling diverse user inputs. The research also highlighted the on-going challenges in content moderation, including bias, context sensitivity, and the difficulty of accurately detecting harmful intent, all of which were considered during system design.

From a project management perspective, the use of an Agile SCRUM methodology allowed me to progress in a structured yet flexible manner. I broke the development into sprints which made it easier to prioritise core functionality and adapt to challenges as they came. Regular iterations made sure that the key components, including the machine learning pipeline, backend API, and frontend interface, were developed incrementally and tested continuously. This approach supported effective time management, although some planned features, such as advanced user analytics, had to be abandoned in favour of delivering a stable core system.

On a personal level, this project significantly strengthened my understanding of full-stack development and machine learning integration. I gained practical experience in training and fine-tuning transformer-based models, handling real-world issues such as class imbalance, and deploying models within a production-like environment. I also developed a deeper understanding of API design, authentication mechanisms, and frontend state management. One key learning outcome was the importance of aligning system components, as challenges came not from individual parts, but from how they interacted, particularly during frontend-backend integration and data handling.

Despite its strengths, the current system has several limitations. The final Figma UI/UX design did not fully match the implemented frontend, primarily because of how the model returned moderation responses and the complexity of state management. Additionally, the model was trained on a pre-existing dataset, which may introduce bias depending on how the data was originally labelled and curated. While the

model performs well overall, performance on minority classes remains slightly lower, reflecting the inherent imbalance in the dataset. The system currently relies on a third-party decoder model (OpenAI) to generate alternative suggestions. In the future I would like to train a decoder model for suggestion generation to remove reliance on external APIs.

Future work will focus on addressing these limitations and extending the system's capabilities. One potential improvement is the development of a dynamic data collection pipeline that continuously gathers and updates training data by scraping platforms such as 4chan and Reddit. This would allow the model to adapt to evolving language patterns, slang, and emerging forms of harmful speech, making it smarter and relevant in real-world scenarios. In the future I'd also like to incorporate advanced user analytics.

In conclusion, I believe this project successfully demonstrates the design, implementation, and evaluation of a machine learning-driven content moderation platform. It highlights both the potential and the limitations of current approaches to automated moderation, while providing a practical solution that integrates modern AI techniques into a real-world application. I believe the project not only achieved its technical objectives but also provided valuable insights into the complexities of building scalable and intelligent systems for content moderation on social media platforms.

9 References

Spectrum Labs. (n.d.). AI-based content moderation: Improving trust & safety online. https://www.spectrumlabsai.com/ai-for-content-moderation/

Kufel, J., Bargieł-Łączek, K., Kocot, S., Koźlik, M., Bartnikowska, W., Janik, M., Czogalik, Ł., Dudek, P., Magiera, M., Lis, A., Paszkiewicz, I., Nawrat, Z., Cebula, M., & Gruszczyńska, K. (2023). What is machine Learning, artificial neural networks and Deep Learning?—Examples of Practical applications in medicine. Diagnostics, 13(15), 2582. https://doi.org/10.3390/diagnostics13152582

Rules - 4chan. (n.d.). https://www.4chan.org/rules#

Kgomo, S. (2025, February 13). I was a content moderator for Facebook. I saw the real cost of outsourcing digital labour. The Guardian. https://www.theguardian.com/commentisfree/2025/feb/12/moderator-facebook-real-cost-outsourcing-digital-labour

Sloan, S. (2021, November 11). The human toll of Facebook's content moderation. The Indy. https://www.theindy.org/article/2522

Michellewrites. (2015, August 21). Facebook: Content creation & Digital Labour. Michellewrites2. https://michellewrites2.wordpress.com/2015/08/20/facebook-content-creation-digital-labour/

Lorenz, T. (2019, September 17). A brief history of internet culture—and how everything became absurd. Medium. https://medium.com/swlh/a-brief-history-of-internet-culture-and-how-everything-became-absurd-6af862e71c94

Ling, J. (2023, June 5). Inside 4chan's Top-Secret moderation machine. WIRED. https://www.wired.com/story/4chan-moderation-buffalo-shooting/

Seijin. (2024, December 13). Social Media Content Moderation: Complete Guide 2025. Enrich Labs. https://www.enrichlabs.ai/blog/social-media-content-moderation-complete-guide

Kurrek, J., Saleem, H. M., & Ruths, D. (2021). Towards a comprehensive taxonomy and large-scale annotated corpus for online misogyny detection. In Proceedings of the 5th Workshop on Online Abuse and Harms (WOAH 2021) (pp. 30–40). Association for Computational Linguistics. https://aclanthology.org/2021.woah-1.3

BERT 101 - state of the art NLP model explained. (n.d.). https://huggingface.co/blog/bert-101

Caselli, T., Basile, V., Mitrović, J., & Granitzer, M. (2021). HateBERT: Retraining BERT for abusive language detection in English. In Proceedings of the 5th Workshop on Online Abuse and Harms (WOAH 2021) (pp. 17–25). Association for Computational Linguistics. https://aclanthology.org/2021.woah-1.3.pdf

Subramanian, M., Sathiskumar, V. E., Deepalakshmi, G., Cho, J., & Manikandan, G. (2023). A survey on hate speech detection and sentiment analysis using machine learning and deep learning models. Alexandria Engineering Journal, 80, 110–121. https://doi.org/10.1016/j.aej.2023.08.038

Horev, R. (2018, November 10). BERT explained: State of the art language model for NLP. Medium. https://medium.com/data-science/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

Kalirane, M. (2025, May 14). Gradient Descent vs. Backpropagation: What's the Difference? Analytics Vidhya. https://www.analyticsvidhya.com/blog/2023/01/gradient-descent-vs-backpropagation-whats-the-difference/

Transformer Explainer: LLM transformer model visually explained. (n.d.). https://poloclub.github.io/transformer-explainer/

GeeksforGeeks. (2025, August 23). Self attention in NLP. GeeksforGeeks. https://www.geeksforgeeks.org/nlp/self-attention-in-nlp/

Smith, B. (2025, January 24). A Complete Guide to BERT with Code. Towards Data Science. https://towardsdatascience.com/a-complete-guide-to-bert-with-code-9f87602e4a11/

Kalirane, M. (2025, May 14). Gradient Descent vs. Backpropagation: What's the Difference? Analytics Vidhya. https://www.analyticsvidhya.com/blog/2023/01/gradient-descent-vs-backpropagation-whats-the-difference/

Why do Reddit and 4chan have such bad user interfaces? (n.d.). Quora. https://www.quora.com/Why-do-Reddit-and-4chan-have-such-bad-user-interfaces

Jeon, S. (n.d.). Streamlining Reddit UI components: The Reddit design system. Medium. https://medium.com/@sehyunjeon/streamlining-reddit-ui-components-the-reddit-design-system-a90189fd9c38

Digilogy. (2025, April 7). Reddit introduces AI-powered community moderation tools. https://news.digilogy.co/reddit-introduces-ai-powered-community-moderation-tools/

Or, B. (2024). Transformer-based Dog Behavior Classification with Motion Sensors. ResearchGate. https://doi.org/10.13140/RG.2.2.21393.40806

Contributors, P. (2023, January 1). BCEWithLogitsLoss. https://docs.pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html

Splitting training data into Train, Validation, and Test Sets. (n.d.). Pecan Help Center. https://help.pecan.ai/en/articles/6454518-splitting-training-data-into-train-validation-and-test-sets

Opinion: Training Argument Fine Tuning MLM RoBERTA - Intermediate - Hugging Face Forums. (2025, January 9). Hugging Face Forums. https://discuss.huggingface.co/t/opinion-training-argument-fine-tuning-mlm-roberta/135013

Zhezherau, A., & Wrike. (2025, September 11). What is the Scrum methodology? | Wrike Scrum guide. Wrike. https://www.wrike.com/scrum-guide/scrum-methodology/

Brena, R. F., Aguileta, A. A., Trejo, L. A., Molino-Minero-Re, E., & Mayora, O. (2020). Choosing the best sensor fusion method: a Machine-Learning approach. Sensors, 20(8), 2350. https://doi.org/10.3390/s20082350

Naderalvojoud, B., & Hernandez-Boussard, T. (2024, January 11). Improving machine learning with ensemble learning on observational healthcare data. https://pmc.ncbi.nlm.nih.gov/articles/PMC10785929/

Lopez, T. (2023, December 1). Simplifying Serialization with Marshmallow: An Introduction with SQLAlchemy. Tiana Lopez. https://tianalopez.hashnode.dev/simplifying-serialization-with-marshmallow-a-guide-to-sqlalchemy