Combatting Game Toxicity with Large Language Models

By

Chia Kwang Yang (A0248658L)

Supervisor: Prof Tong Xin

DSA4288 Honours Project in Data Science and Analytics

Department of Statistics and Data Science

National University of Singapore

2024/2025

Contents

Abstract

Toxic behavior in online multiplayer games continues to be a significant concern, affecting player experience, community integrity, and game longevity. While traditional approaches to toxicity detection have relied on supervised machine learning or manually engineered features, recent advancements in natural language processing offer a new frontier. This thesis presents a novel pipeline that leverages decoder-based Large Language Models (LLMs), specifically the Llama 3.1-8b-Instruct model, to detect toxic behavior in multiplayer game chat logs. The proposed system uses a two-stage LLM approach: (1) keyword extraction to identify game-specific jargon, meme terms, slang, and non-standard English expressions, and (2) toxicity prediction based on message content enriched by context-sensitive information retrieved from a custom-built vector database.

The core innovation lies in our Retrieval-Augmented Generation (RAG) strategy, specifically a new variant called RAG-Keyword, which improves upon RAG-Token and RAG-Sequence methods by only retrieving semantic definitions for meaningful keywords. This approach preserves efficiency while enhancing interpretability and accuracy. The retrieval process is further optimized by incorporating a hybrid similarity metric that blends cosine similarity and a normalized Levenshtein distance to account for the misspellings and abbreviations common in high-speed gaming chat. This system allows the LLM to better contextualize unfamiliar or obfuscated toxic terms before making classification decisions. We use Pinecone to structure our vector database into namespaces segmented by game and region, enabling localization of meaning for slang and jargon, especially between Southeast Asia and Oceania.

To evaluate the pipeline, a dataset of 150 real-world chat messages collected from four games. Counter-Strike, League of Legends, Teamfight Tactics, and Marvel Rivals data were annotated and used to test the model. The system achieves a Jaccard Similarity of 0.687 for

keyword detection, demonstrating high fidelity with human annotations. For toxicity prediction, incorporating RAG led to a notable increase in recall for toxic messages from 0.86 to 0.93, and an F1-score improvement from 0.72 to 0.74 for the minority toxic class, while maintaining strong performance on non-toxic samples. These improvements are critical in real-world settings where missing toxic messages carries high social cost.

Additionally, the system incorporates Explainable AI (XAI) principles by requiring every toxicity prediction to include a human-readable rationale, thereby aiding in moderation decisions and increasing user trust. The use of few-shot prompting, lightweight models, and modular pipeline components ensures scalability, cost-effectiveness, and adaptability across different gaming platforms and regions.

This work contributes not only a functional prototype for real-time toxicity detection, but also introduces a scalable and linguistically robust framework that balances accuracy, interpretability, and resource constraints. Future improvements include dynamically identifying new keywords with low similarity scores by searching social media for context and using LLMs to extract definitions. The system can also be extended to voice data via speech-to-text and enhanced with batch message processing for improved accuracy, though both were limited by data constraints in this project. Ultimately, this project takes into account the linguistically unique issues and limitations of gaming, and utilises well-thought out methods to addresses these issues and limitations to form LLM-based solutions.

1 Introduction

Disclaimer: This paper consists of vulgar language and sexual representations in text. Such content involved is used as examples to explain concepts or as criteria for our objectives.

Toxicity in online gaming refers to behaviours such as harassment, hate speech, and disruptive conduct that negatively impact players' experiences. (Robertson, 2020) It has been a problem ever since the introduction of competitive multiplayer gaming. Additionally, toxicity can create unwelcoming environments, discouraging participation from women and minority groups, and potentially affecting their mental health (Robertson, 2020). Moreover, there have been instances whereby players were unfairly penalized due to misinterpretations. For example, players in League of Legends have reported suspensions for toxicity despite not being engaged in toxic behaviour, highlighting the challenges in these automated moderation systems (Unbanster, n.d.).

In this paper, we discuss utilising Retrieval-Augmented Generation (RAG) in this unique use case with a few considerations in order to better toxicity detection systems that are implemented in different multiplayer games:

1. Gamers often make more mistakes and abbreviate more words due to the fast-paced nature of in-game communication. Gamers also often try to obfuscate messages in efforts to dodge detection. Therefore, we introduce alternative objectives for us to account for such phenomena.

Figure 1: Toxic conversation between users “riival” and “aiukhgiang”. “aiukhgiang” obfuscates their messages by adding spaces between the letters “fuc” and “k”.

2. Toxicity is often hard to catch in different contexts, with different societies using words differently. Sometimes, even with the words that exist within linguistically similar societies (e.g. Australia & the United States), different words can be used in different settings. One such example is use of the word “cunt”, whereby Australians often use the word in casual settings while Americans would regard the word as highly offensive.

3. Lastly, we consider that game toxicity is often reported, but the report is not immediately acted on. Therefore, we disregard the speed needed during RAG and focus solely on the metrics of better toxicity predictions.

2 Literature Review

In this section, 2 papers and their methodologies to detect toxicity in multiplayer gaming are reviewed. Firstly, a paper in 2014 called “STFU NOOB!: Predicting crowdsourced decisions on toxic behaviour in online games behaviour in online games” by Jeremy Blackburn in the University of South Florida and Haewoon Kwak from Telefonica Research is reviewed. They utilise the now-defunct League of Legends (LoL) Tribunal, which is a crowdsourcing system to judge whether reported players should be punished (Blackburn et al., 2014). The paper explores in-game behavioural features such as player performance metrics, chat logs, and user reports to develop a Random Forest classifier for toxicity prediction. The data that was utilised is a mixture of textual data (such as user reports and chat logs) as well as tabular data (such as in-game performance statistics) (Blackburn & Kwak, 2014), which is out of the scope for this project. Moreover, the use of traditional machine learning techniques disregards the use of new vocabulary that will be used in text, and therefore would require a lot of retraining every time new game-specific jargon appears. Depending on the frequency of updates which leads to this game-specific jargon, this could potentially lead to frequent retraining, which will be a huge waste of resources. Regardless, the literature review of this paper shows that traditional machine learning can be used to predict game toxicity and achieve high results.

Secondly, a paper in 2024 called “Game On, Hate Off: A Study of Toxicity in Online Multiplayer Environments” by Yang (2024) was reviewed. The paper investigates toxicity trends in multiplayer games by analysing 8 months of chat data from two Ubisoft games, For Honor and Rainbow Six Siege (Yang, 2024). The paper utilises millions of chat messages annotated using a structured framework for toxic content, distinguishing between different types of harmful speech including hate speech, threats and harassment. The paper implements a RoBERTa-base transformer model that is finetuned on labelled toxicity datasets from

Ubisoft's games (Yang, 2024). The RoBERTa-base transformer model utilises the encoder-decoder architecture of the proposed transformer model in "Attention Is All You Need". The model is first pre-trained with unlabelled chat data with the objective of masked language modelling, where a model predicts masked or hidden words in a sentence with respect to the surrounding context. They then fine-tune on labelled toxicity data categorized into different severities, defining the problem with the objective of multi-class classification. This project differs from the second paper in a few ways. Firstly, we do not train a model on our own and instead utilise pre-trained large language models (LLMs). Therefore, while the paper's defined problem might be the same as this project, our project does not utilise supervised or unsupervised training methods to develop a model from scratch. Instead, our project leverages pre-trained large language models and utilises Retrieval-Augmented Generation as a prompt engineering method to further improve prediction accuracy. Secondly, the model architecture is different from the models used in our project. As mentioned, the RoBERTa-based transformer model utilises the encoder-decoder architecture. However, the models that are being used for this project are transformer models that utilise the decoder architecture. They differ in the parts of the transformer that are being activated. The RoBERTa-based transformer model utilises the whole transformer architecture, while decoder-based transformer models only use the latter half of the transformer architecture. Regardless, the literature review of this paper shows that transformer-based models are feasible for toxicity classification, and therefore we will use decoder-based models for our approach.

3 Theory: Encoder-Decoder Transformers VS Decoder-only Transformers

In this section, we discuss the difference between encoder-decoder transformers and decoder-only transformers. The original transformer consists of an encoder, which takes in the input sequence which are converted to embeddings, before combining with positional encodings and feeding it through a feed-forward network consisting of Multi-Head Attention layers (Vaswani et al., 2017). The decoder layer first processes the input token-by-token, and alongside the positional encoders, feeds both through a masked multi-head attention layer before another multi-head attention layer, finally ending off with a feed forward network. For models like BERT and RoBERTa, where it utilises both the encoder and decoder, the multi-head attention layer takes in input from the encoder. Moreover, the input for encoders and decoders for that architecture will not be the same. They are usually inputs that are shifted 1 step forward. The architecture has been given below:

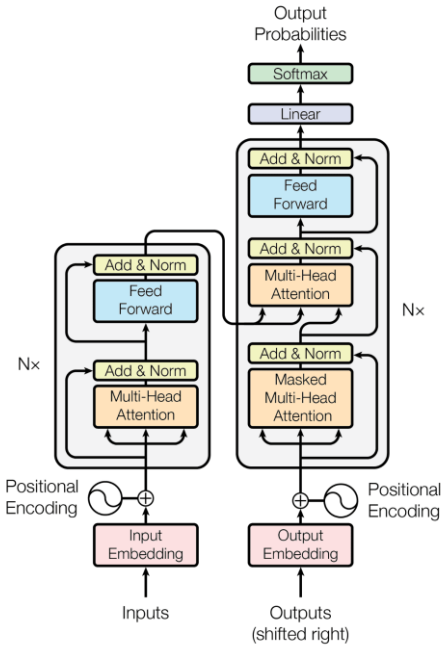

Figure 2: Transformer encoder-decoder architecture. Source: Vaswani et al., 2017

However, for decoder-based models only such as GPT, Claude, and Llama, it does not include input from encoders, and the model only relies on self-attention to process its input and generate output. The input for decoders is also not shifted a step forward, as compared to those of the encoder-decoder architecture.

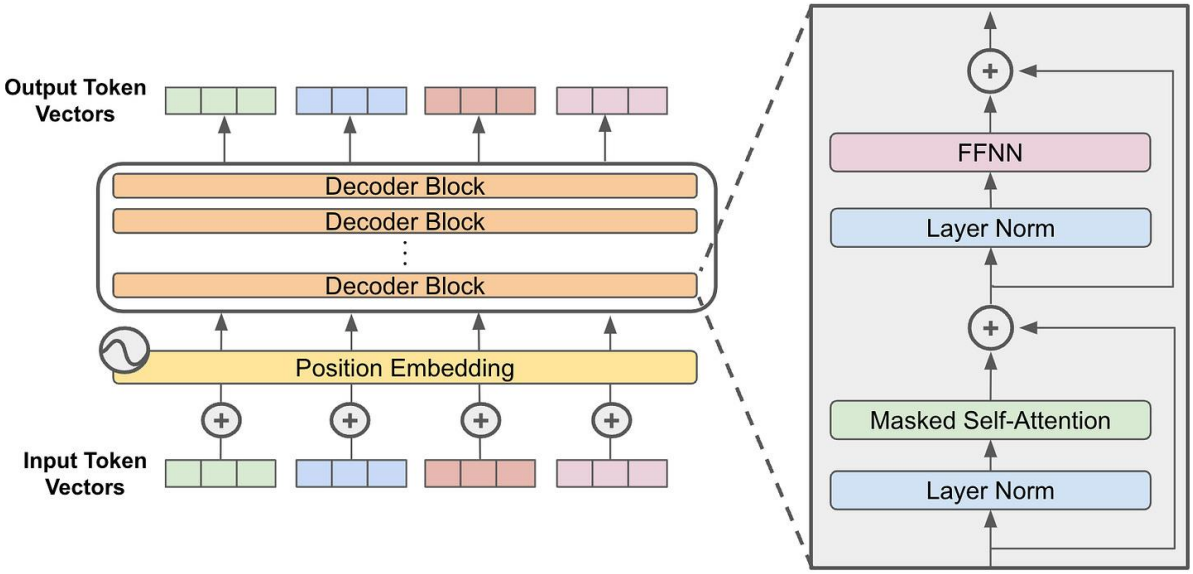

Figure 3: Transformer decoder architecture. Source: Wolfe, C. (2023). Decoder-Only Transformers: The Workhorse of Generative AI. Retrieved from https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse

While transformer models that utilise encoder models are generally more effective in classification-based objectives, given different tricks such as few-shot learning as well as prompt engineering, we are able to improve the performance of a decoder-based model to be as effective as models utilising encoders in classification-based objectives. We use few-shot learning in our prompts for our prediction.

4 Process

In this section, we talk about the mathematics behind the RAG mechanism, as well as the adjustments made to normal RAG systems to better accommodate our problem.

4.1 Overall Process

Our aim is to simulate an Application Programming Interface (API) that is developed to predict game toxicity. Once a player has been reported in-game by other players (of which we will call the offender), the API will be called. The offender's messages will then be sent through the API to extract the keywords from the message itself. We use LLMs for the task of keyword extraction as we will need to extract out words, specifically game-specific jargon, that the model does not recognise. These keywords will be then queried into a vector database, consisting of the embeddings and the metadata of the game-specific jargon. As players might not type the correct word due to mistakes made in the heat of the moment, we also include word edit distance as a metric to determine the similarity of the word. We create the vector database on our own as there is currently no available market option that supports this configuration. Lastly, we add the metadata of the word most similar to the keyword into another prompt that is fed into a new instance of a model to determine whether the offender is toxic.

Secondly, we use LLMs for the task of toxicity prediction, whereby we prompt engineer our query to predict whether the messages are toxic or not. If the offender has been deemed to be toxic in-game, we will return both the result and the offending messages that have been written during the game itself. If the offender has otherwise not been deemed toxic during the game, we will only return the result itself. To achieve this aim, we build a pipeline that consists of LLMs to achieve two tasks. While we do not have the exact format of the data being sent through games, we simulate the data being sent with the fields 'user', 'message' and 'game' for every message in a game session.

4.2 Keyword Extraction

We first utilise LLMs to extract the key words of a message itself. With the data that we have available, we establish the following rules to extract said key words:

1. **Common Gaming Phrases** - frequently used expressions in online multiplayer games that reflect sportsmanship or game status.
   Example: "gg", "gl", "hf", "nt", "ez", "lol", "wp"
2. **Game-Specific Actions** - terms that describe in-game activities or player roles commonly encountered during gameplay.
   Example: "planting", "defusing", "save", "heal", "heals", "dps"
3. **Character or Ability Names** - names of in-game characters or special abilities unique to certain games.
   Example: "spiderman", "psylocke", "hawkeye", "jeff"
4. **Internet Slang and Meme Words** - informal expressions or meme-related terms that have gained popularity in gaming and online communities.
   Example: "goat", "lmao", "yawn", "noob"
5. **Non-Standard English Terms** - words that are not typically found in standard English dictionaries and often reflect regional slang or foreign expressions.
   Example: "chao", "kaopei", "basah"

The keywords are not usually repeated per message as it is typically not common behaviour from gamers. However, in the case of such an event, the keywords that have been repeated will be preprocessed by the LLM to exclude any duplicates.

All keywords must be extracted without applying stemming or lemmatization (e.g. "heal" and "heals" are treated as distinct terms). The matching process is case-insensitive, and all punctuation should be stripped from the message before keyword comparison. If a chat

message contains no keywords from the whitelist, the result should be an empty list. The final output must be formatted in JSON, with each object containing two keys: "message", representing the original chat message, and "keywords", representing a list of matched keywords.

4.3 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) is used as a technique to provide more context for the LLM. It retrieves a document for each sequence of words (more commonly known as query to the layman), or for each token in the query itself. It retrieves the top-k documents (where k is defined by the user) and uses these retrieved documents as additional context in order to generate more accurate, relevant and factually grounded responses. Retrieval-Augmented Generation (RAG) is defined as two formulas in "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks", one for a sequence of words (known as RAG-Sequence) and one for each token in the query itself (known as RAG-Token). We will introduce both models in our project, but will also discuss the strengths and weaknesses of both models. We will then introduce our new model, RAG-Keyword, that utilises the strengths of both models for our use case.

4.3.1 RAG-Token Model

The probability of generating the output sequence y given the input x using RAG-Token is approximated by the product of the sum of the retriever probability $p_\eta(z \mid x)$ multiplied by the generator probability $p_\theta(y_i \mid x, z, y_{1:i-1})$ for each individual token, from the first to the Nth token in y. This is expressed as:

$$p_{\text{RAG-Token}}(y \mid x) \approx \prod_{i=1}^{N} \sum_{z \in \text{top-}k(p(\cdot \mid x))} p_\eta(z \mid x)\, p_\theta(y_i \mid x, z, y_{1:i-1})$$

The retriever probability $p_\eta(z \mid x)$ is defined as the probability assigned by the retriever (with parameters η) of selecting document z given the input x.

The generator probability $p_\theta(y_i \mid x, z, y_{1:i-1})$ is defined as the probability assigned by the generator (with parameters θ) for producing the individual token y_i, conditioned on the input x, the retrieved document z, and all previously generated tokens $y_1, y_2, \ldots, y_{i-1}$.

On a practical basis, we use an example to denote the RAG-Token model. Given the example and the LLMs that we utilise, we realise that the model will break down the whole word to even smaller sub-word tokens instead of typical processing that takes in each word as a token. Let us use the example “bro ur blind he was at stairs???” to process the whole model.

Therefore, the steps are as such:

1. We first need to preprocess the text data into tokens.Therefore, for the sentence above, this is broken into sub-word tokens, i.e. ["bro", "ur", "blind", "he", "was", "at", "st", "airs", "?", "?", "?"]
2. Each of those tokens will be converted into embeddings before it independently triggers a document retrieval from a vector database. Even super small subword tokens such as “ur” and “?” will trigger the document retrieval. For our case, the documents that we retrieve are meanings of the tokens themselves.
3. After that, the model will combine the input, the token-specific retrieved document as well as its previous outputs. In our case, there is no previous output as every time we call the model, we initialise a new instance of the model.

The flow of the model is visualised as such:

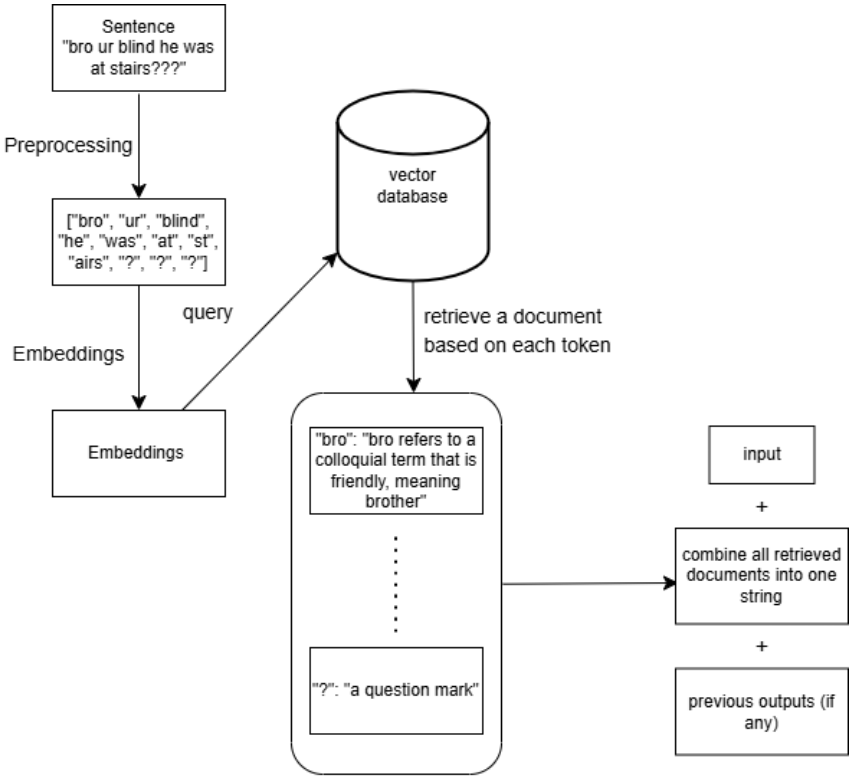

Figure 4: RAG-Token Model Workflow Visualisation

However, there are a few problems with the RAG-Token model. Firstly, retrieving a document for each token, especially for LLM-based preprocessing, will also include super small sub-word tokens, as well as semantically weak sub-words. Therefore, this leads to unnecessary retrieval overhead, and potential context noise.

Secondly, some gaming terms are not handled correctly during the model's preprocessing phase. Since large language models apply generic tokenization and normalization rules, certain game-specific jargon may be improperly split, altered, or overlooked. This leads to inaccurate interpretation or missed context during inference.

Thirdly, RAG-Token treats all tokens equally during retrieval, without considering their relative importance within the input. In practice, not all tokens contribute equally to the intended meaning of a query. For example, retrieving documents for generic tokens like “have” or “does” offers little value compared to retrieving for terms like “Psylocke” or “ultimate ability.” This uniform treatment prevents the model from prioritizing contextually important tokens, which can dilute the relevance of retrieved information and reduce prediction quality.

4.3.2 RAG-Sequence Model

To provide an alternative model for RAG, we present the **RAG-Sequence Model**, which retrieves only one document for each sequence of text. The RAG-Sequence model is defined mathematically as follows:

$$p_{\text{RAG-Sequence}}(y \mid x) \approx \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z \mid x)\, p_\theta(y \mid x, z)$$

$$= \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z \mid x) \prod_{i=1}^{N} p_\theta(y_i \mid x, z, y_{1:i-1})$$

The probability of generating the output sequence y given the input x using RAG-Sequence is approximated by the sum of the retriever probability $p_\eta(z \mid x)$ multiplied by the generator probability $p_\theta(y \mid x, z)$ for each retrieved document z from the top-k most relevant documents.

The generator probability $p_\theta(y \mid x, z)$ is further decomposed into the product of token-level probabilities from the first to the Nth token in y. Each token y_i is generated based on the input x, the retrieved document z, and all previously generated tokens $y_1, \ldots, y_{i-1}$.

Meanwhile, the retriever probability $p_\eta(z \mid x)$ is defined as the probability assigned by the retriever (with parameters η) of selecting document z given the input x.

Once again, we use the example “bro ur blind he was at stairs???” to process the whole model. Therefore, the steps are as such:

1. We first need to preprocess the text data into tokens. Therefore, for the sentence above, this is broken into sub-word tokens, i.e. ["bro", "ur", "blind", "he", "was", "at", "st", "airs", "?", "?", "?"].
2. Unlike RAG-Token, RAG-Sequence performs a single retrieval for the entire input query. The input sentence is first converted into a single embedding, and the top-k most

relevant documents are retrieved based on this full sentence embedding from the vector database. In our case, the document that we retrieve are from meanings of the keywords likely present in the sentence.

3. After that, the model uses the same retrieved document(s) as context for generating all tokens in the output sequence. Each token is generated based on the original input, the shared retrieved context, and previously generated tokens. However, in our case, there is no previous output as every time we call the model, we initialise a new instance of the model. Therefore, all tokens are generated in one go using the same context.

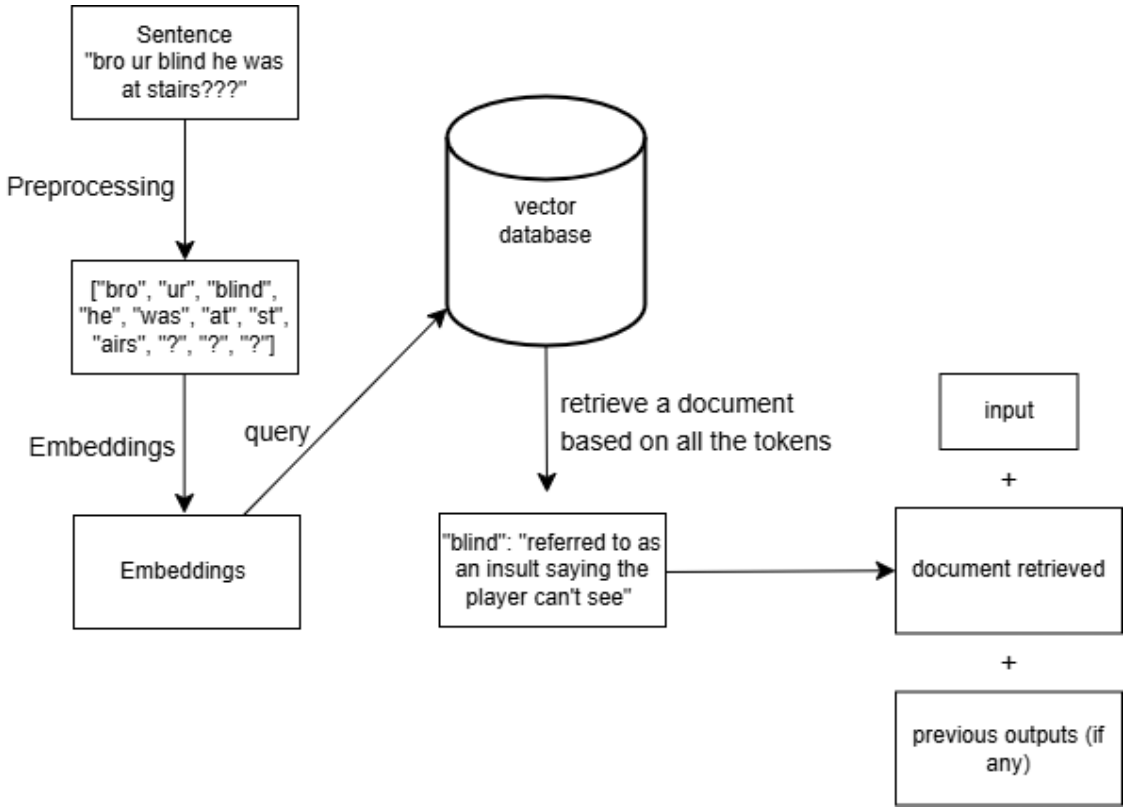

Figure 5: RAG-Sequence Model Workflow Visualisation

The RAG-Sequence model does solve some of the problems detailed in the previous section. Since it only calls the vector database once, this will not lead to unnecessary retrieval overhead and reduce potential context noise. However, it creates new problems that is inappropriate for this use case.

Firstly, while it results in improved efficiency, it is insufficient for our use case. Most of the text in toxic messages use game-specific or region-specific jargon, and therefore one retrieval will limit the model's ability to incorporate the diverse contextual information needed to accurately interpret such terms.

It also does not solve the problem of generic tokenization and normalization rules and does not consider its relative importance within the input itself. In fact, it overvalues the specific message's semantic meaning when for this problem, it would be much more effective to get the meaning of keywords not understood by the LLM.

Therefore, we develop the RAG-Keyword Model for us combine the strengths of both models, to provide us a model that would fit our use case best.

4.3.3 RAG-Keyword Model

Since for our approach, we do not need to retrieve documents for every single token in the query itself, therefore we slightly adjust the definition to better suit our objective.

RAG-Keyword utilises the same mechanism as RAG-Token, but we do not call on RAG for each token, instead only on a subset of words in the query deemed keywords.

Therefore, we define the **RAG-Keyword model** as follows: Given input x, we have a generated sequence $y = (y_1, y_2, \dots, y_N)$. Assume a set of indices $J \subseteq \{1, \dots, N\}$ corresponding to the positions of identified keywords. Then the probability of generating the output sequence y given input x is approximated as:

$$p_{\text{RAG-Keyword}}(y \mid x) \approx \prod_{i=1}^{N} \begin{cases} \sum_{z \in \text{top-}k(p(\cdot \mid x, y_i))} p_\eta(z \mid x, y_i)\, p_\theta(y_i \mid x, z, y_{1:i-1}), & \text{if } i \in J \\ p_\theta(y_i \mid x, y_{1:i-1}), & \text{otherwise} \end{cases}$$

The probability of generating the output sequence y using the RAG-Keyword model is approximated by multiplying probabilities across each token from the first to the Nth token. These probabilities are defined by a piecewise function:

- **If the token is a keyword** (i.e., $i \in J$): we sum over the top retrieved documents relevant to the keyword y_i. The final probability is the product of the retriever probability $p_\eta(z \mid x, y_i)$, which represents the likelihood of selecting document z given both x and the keyword y_i, and the generator probability $p_\theta(y_i \mid x, z, y_{1:i-1})$, which represents the likelihood of generating token y_i given the context.

- **If the token is not a keyword** (i.e., $i \notin J$): no retrieval is performed. We directly use the generator probability $p_\theta(y_i \mid x, y_{1:i-1})$, computed without any external context.

The final sequence probability is obtained by multiplying all token-level probabilities together.

We once again use the example “bro ur blind he was at stairs???” as an example of using RAG-Keyword. In this case, we assume that the keywords retrieved by the LLM is “blind” and “stairs”. Therefore, the steps are as such:

1. We first preprocess the text data into words instead of tokens. Therefore, for the sentence above, this is broken into words, i.e. ["bro", "ur", "blind", "he", "was", "at", "stairs", "???"].
2. Instead of retrieving a document for every token, we use a keyword extraction model to identify which words are meaningful and relevant to our use. The criteria are defined in the section 4.2 above. Only the words identified as keywords will trigger a document retrieval from the vector database. Words not identified as keywords (e.g., “he”, “was”, “???”, etc.) will not initiate any retrieval. In our case, the documents retrieved are definitions or contextual meanings of the identified keywords.
3. After that, the model will combine the input, the keyword-specific retrieved documents, and its previously generated outputs to generate each token. For non-keywords, generation proceeds without any retrieval. In our case, there is no previous output as every time we call the model, we initialise a new instance of the model. Therefore, all tokens are generated independently using the appropriate context.

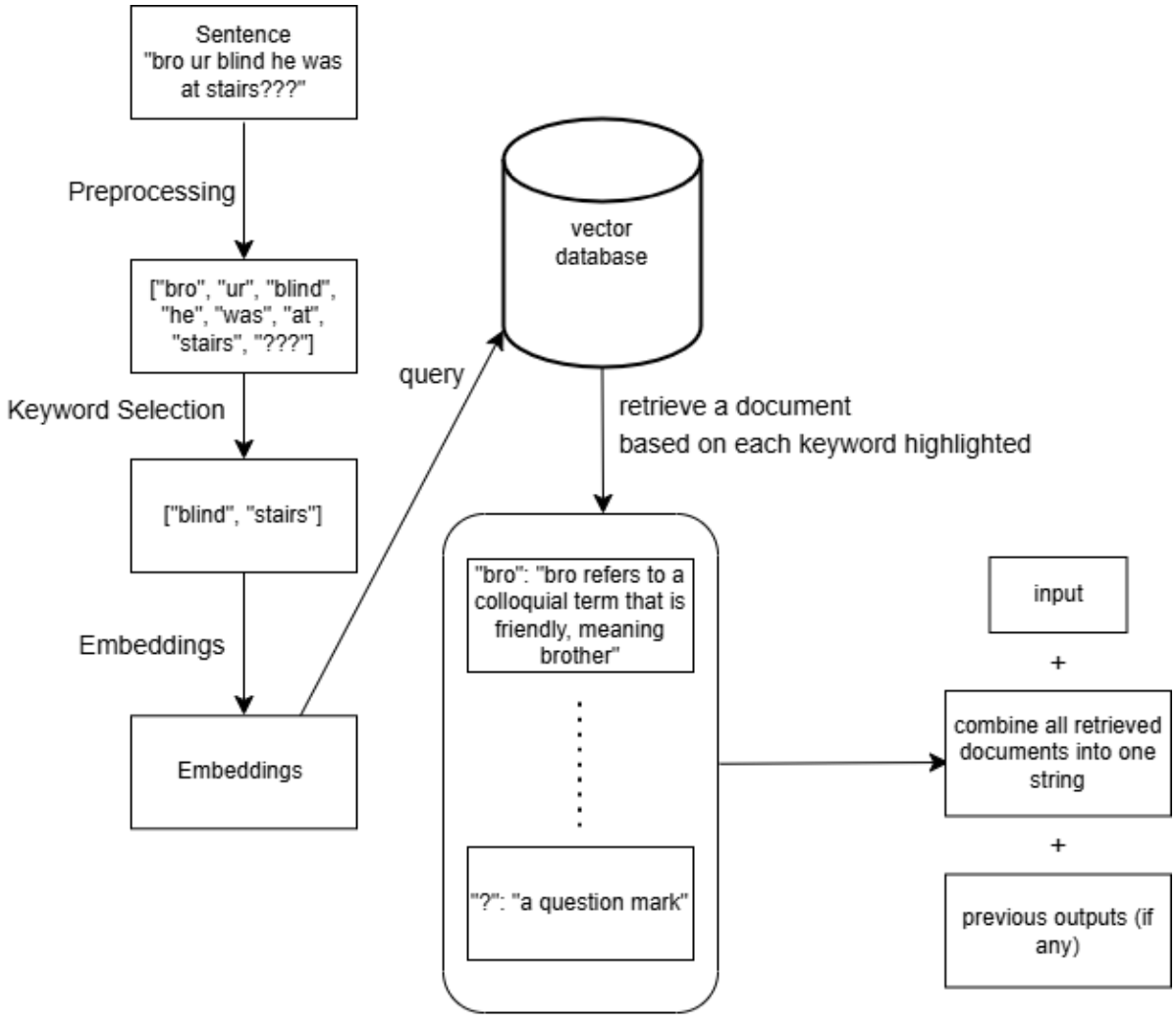

Figure 6: Proposed RAG-Keyword Workflow Visualisation

The methodology is still the same as RAG-Token with the slight exception of retrieving the document if the word is deemed a keyword. However, it combines the strengths of RAG-Token and RAG-Sequence whereby it cuts down on the calling costs, while making sure that there is still enough context for the query itself. It also solves the other two problems mentioned in the RAG-Token section, whereby the tokenization and normalization rules are not used since it specifies based on the words themselves. Lastly, there is a weight to the keywords and therefore leads to the consideration of relative importance within the input.

4.3.4 Custom Scoring Objective For RAG-Keyword

In this section, we discuss the new objective introduced into RAG-Keyword. As mentioned in 3.3.3, RAG will take the document that is best suited to the keyword itself. Let us mathematically define RAG and showcase changes to it.

Let $f_{\text{enc}}(q)$ be the query embedding for the original query q, $f_{\text{enc}}(c_i)$ be the embedding of chunk c_i, and c^* be the most relevant chunk with the highest cosine similarity:

$$c^* = \underset{c_i \in \mathcal{C}}{\text{argmax}} \frac{f_{\text{enc}}(q) \cdot f_{\text{enc}}(c_i)}{\| f_{\text{enc}}(q) \| \| f_{\text{enc}}(c_i) \|}$$

We will then concatenate c^* with the user's query into the LLM. Let f_{LLM} be the large language model, and a be the generated response:

$$a = f_{\text{LLM}}(\text{concat}(c^*, q))$$

The concatenation function is defined as:

$$\text{concat}(c^*, q) = \{c_1^*, c_2^*, \dots, c_m^*, q_1, q_2, \dots, q_n\}$$

Thus, the full RAG process can be rewritten as:

$$a = f_{\text{LLM}}\left(\text{concat}\left(\underset{c_i \in \mathcal{C}}{\text{argmax}} \frac{f_{\text{enc}}(q) \cdot f_{\text{enc}}(c_i)}{\| f_{\text{enc}}(q) \| \| f_{\text{enc}}(c_i) \|}, q\right)\right)$$

However, for our use case, we recognise that gamers often type rapidly, introducing spelling mistakes. This alters the semantic meaning of words, potentially leading to incorrect keyword definition retrieval. To mitigate this, we introduce Levenshtein distance as an additional objective to minimize. It is defined recursively as:

$$d(i,j) = \begin{cases} \max(i,j), & \text{if } \min(i,j) = 0 \\ \min \begin{cases} d(i-1,j)+1 \\ d(i,j-1)+1 \\ d(i-1,j-1)+\delta(a_i,b_j) \end{cases}, & \text{otherwise} \end{cases}$$

To normalize and align with the [0, 1] range of cosine similarity, we define Levenshtein similarity as:

$$\text{sim}_{\text{lev}}(q,c_i) = 1 - \frac{d(q,c_i)}{\max(|q|,|c_i|)}$$

We then combine cosine similarity and Levenshtein similarity into a unified score:

$$\text{score}(q,c_i) = \alpha \cdot \frac{f_{\text{enc}}(q) \cdot f_{\text{enc}}(c_i)}{\| f_{\text{enc}}(q) \| \| f_{\text{enc}}(c_i) \|} + (1-\alpha) \cdot \left(1 - \frac{d(q,c_i)}{\max(|q|,|c_i|)}\right)$$

Finally, we redefine the full RAG process with this hybrid score:

$$a = f_{\text{LLM}}\left(\text{concat}\left(\underset{c_i \in \mathcal{C}}{\text{argmax}} \text{score}(q,c_i), q\right)\right)$$

For our use case, we assign a coefficient $\alpha = 0.7$ to the cosine similarity (which captures semantic relevance) and $1 - \alpha = 0.3$ to the Levenshtein similarity (which compensates for typographical variations typical in fast-paced game chats).

4.3.5 Optimization Processes

For RAG, we consider a few optimization processes in order to accommodate constraints as well as to optimize the speed for this process.

Firstly, we consider that with the increase in the scale of the vector database, we are not able to query every single vector stored in the vector database. Therefore, we utilise the nearest-neighbours technique in order to find the top-k vectors that have the highest cosine similarity to the word itself.

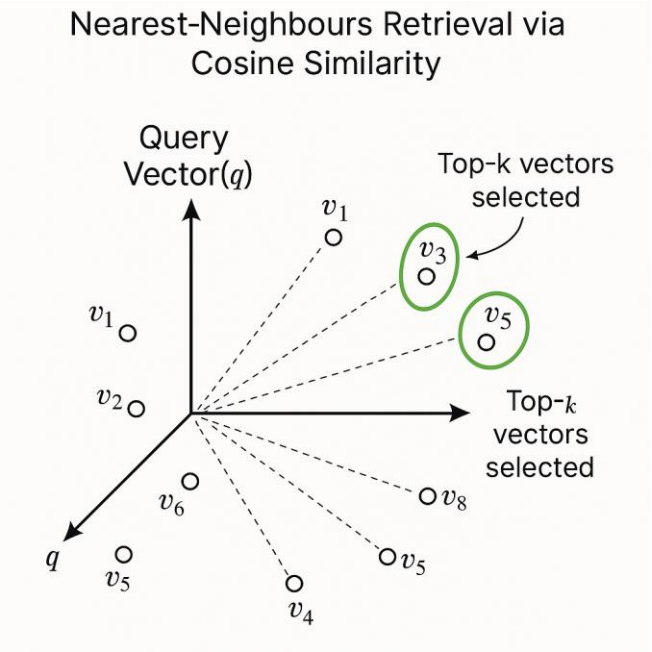

Figure 7: Nearest-Neighbours Retrieval with Cosine Similarity.

Nearest-Neighbours Retrieval is mathematically defined as follows: Let $q \in \mathbb{R}^d$ be the query vector and let $\mathcal{C} = \{c_1, c_2, \dots, c_n\} \subset \mathbb{R}^d$ be the set of candidate vectors in the vector database. We define $\cos(q, c_i)$ as the cosine similarity between q and c_i, the exact formulation of which is provided in Section 4.3.4.

Then, the top-k nearest neighbours of query q are defined by:

$$\mathrm{NN}_k(q) = \arg\max_{S \subset \mathcal{C}, |S|=k} \sum_{c \in S} \cos(q, c)$$

This formulation seeks the subset S of $\mathcal{C}$ with exactly k elements that maximize the total cosine similarity with the query vector q. In practice, this is used to identify the most semantically relevant vectors for retrieval-augmented generation tasks.

Secondly, given the infrastructure that is optimised for vector retrieval, we are not able to calculate the Levenshtein distance alongside the cosine similarity. Therefore, we first calculate the cosine similarity between the keywords and the vectors. We get the top 2k vectors (instead of k vectors) and calculate the Levenshtein distances for each of the words associated with the vectors. We then rerank them to fulfil both the cosine similarity and Levenshtein distance objectives, to find the best word that fulfils both objectives.

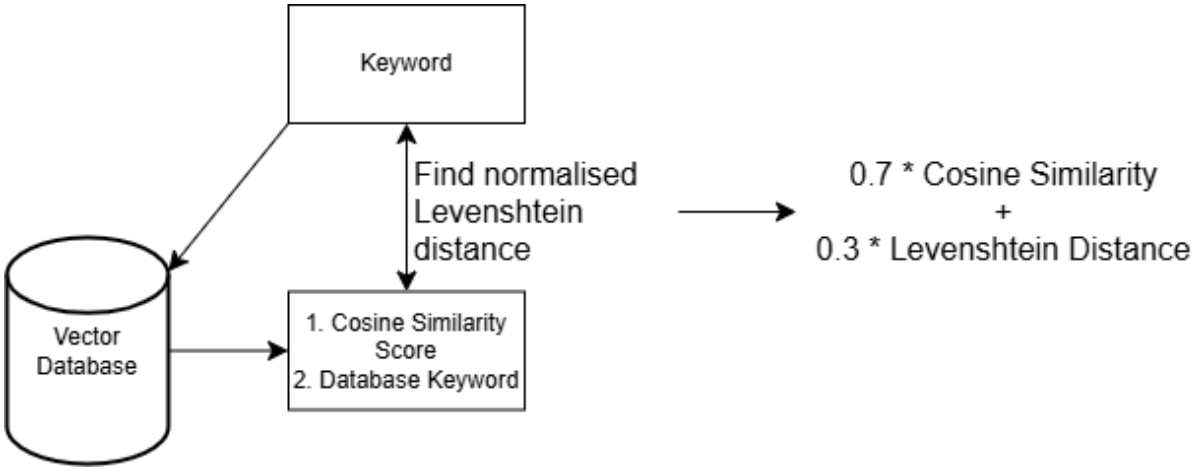

Figure 8: Process of finding the custom scoring in practice

4.4 Prediction

Lastly, we utilise a LLM to predict whether the message is toxic or not. While there are different definitions of toxicity, for this thesis, we set these rules to define as toxic:

1. **Profanities that are aggressive to others** - vulgar expressions intended to insult or provoke another person.

   Example: "fuck you", "stfu", "suck a dick"
2. **Hate speech or slurs** - offensive language targeting someone's race, ethnicity, nationality, religion, or identity.

   Example: "china man", and other racial or religious slurs
3. **Personal insults** - direct attacks on a person's abilities, intelligence, or worth, often used to demean them during gameplay.

   Example: "noob", "low iq", "get a life"
4. **Sexual harassment or violent remarks** - sexually explicit or threatening messages intended to intimidate or offend.

   Example: "suck a dick", “your mum got owned last night?”
5. **Psychological attacks or ableist slurs** - comments that mock mental health, cognitive abilities, or disabilities.

   Example: "brain disorder", "retard"

With those criteria, if a message contains any of these toxic elements, it must be labelled as **toxic**. If the message does **not** contain any of these elements, it must be labelled as **non-toxic**, regardless of sarcasm, annoyance, or mild frustration. The final output must be formatted in JSON, with each object containing two keys: "message", representing the original chat message, and "toxicity", a Boolean value indicating whether the message is toxic (true) or non-toxic (false).

5 Approach

In this section, we detail the approach taken to create the best system to process our problems. Both keyword detection and toxicity prediction utilise LLMs in their process, and in our example, we use a Llama-3.1-8b model for keyword detection and toxicity prediction.

5.1 Assumptions

There are a few assumptions that we have made throughout this process when designing our approach, namely that:

- Given the use of RAG in our system, we do not need to have a very powerful model to make predictions
- Reports of toxicity is not expected to be followed up immediately, it can take time to do so
- We want to cut the amount of money needed to process these predictions; therefore, we should not have a model that has too many parameters, which leads to higher costs.

Therefore, we have chosen the Llama-3.1-8b-Instruct model for both keyword detection and toxicity prediction. The Llama-3.1-8b model follows a decoder-only model architecture and is trained on much lesser parameters than some of the stronger models by other companies such as OpenAI and Anthropic, and even its own company (Meta). Therefore, this model is not only capable enough for such tasks, but also cheap enough for the model to scale as it only costs $0.30 per million input tokens and $0.61 per million output tokens when hosted on Azure (PromptHub, 2024) as compared to larger and more performant models such as OpenAI's GPT-4o at $2.50 per million input tokens and $10 per million output tokens (OpenAI, 2025). It should also be noted that only toxicity prediction uses RAG as an additional system to improve its performance, as keyword detection is part of the step that leads to the RAG mechanism being activated in the overall flow.

5.2 Dataset

The dataset consists of 33 instances of games, with 150 messages altogether from the games themselves. These messages are taken from 4 games, namely Counter-Strike, League of Legends, Teamfight Tactics and Marvel Rivals. The columns of the dataset are as follows:

- **user**: The username of the individual who sent the message.
- **message**: The content of the message that was sent.
- **game**: The specific game in which the message was sent.
- **toxicity**: A score or label indicating the level of toxicity in the message.
- **keywords**: Key terms or phrases extracted from the message.
- **location**: The region of the game played (e.g. Southeast Asia, Australia)

These datapoints were collected from real-time game matches, with interactions from real players. These datapoints are collected from two regions, mainly Singapore as well as Oceania (in this case, Australia only). These datapoints were collected from 1st January 2025, until 6th April 2025. Out of the 150 messages, 128 messages were manually labelled to not be toxic, while 22 messages were manually labelled to be toxic. The dataset consists of a few non-English languages, such as Vietnamese and Bahasa Melayu. Some of the messages consisted a few ways to mitigate the current guardrails of the games such as changing certain letters of offensive words, of which the Large Language Model processes them into actual words instead of its hidden form.

Each instance of conversation is first converted into a JSON formal with the above columns as keys. We then process each conversation through a script “process.py” before combining it into a dataset named “processed_data.csv”. We then use some columns of each datapoint to feed into the LLM and the vector database itself.

5.3 Keyword Detection

For keyword detection, we merely use the LLM without any RAG. We first prompt the model with the prompt consisting of a few examples for few-shot learning in "keyword.txt" (Appendix A). We then feed it through the model to give us the JSON with both the "message" and "keywords" keys. We take the "keywords" keys and compare it to the processed_data.csv with the Jaccard Similarity metric.

5.4 Vector Database

To perform RAG, we will need to use a vector database to perform RAG. We utilise Pinecone for RAG due to the capabilities needed for our project. Firstly, as mentioned before, our data consists of a few columns that we keep, simulating real-world data. These columns help us with finding the semantic meaning of the messages, to more accurately determine whether these messages are truly toxic or not as certain regions and games might use words differently. Pinecone allows us to do this easily as its vector databases allow us to further separate vector databases into different namespaces, akin to tables in normal SQL databases. We use .csv files that have the columns "word" and "meaning" (Appendix B). The names of the files themselves are the namespaces of the vector database itself. We also have a general namespace that stores game-specific jargon that is not used uniquely in any game. With this, every time we classify a message, we also take the region and the game and call the general namespace, which consists of words that are universal in the context of gaming. This allows us to call the relevant namespaces of the region and game of the message itself. Based on that, we can get the semantic meanings of the keywords from the message more efficiently.

5.5 Toxicity Prediction

For the toxicity prediction model, we populate it with the prompt in prompt.txt (Appendix A). There are two ways whereby we use the model. Firstly, we utilise the model as is, without the use of RAG in our toxicity prediction. This is to provide a baseline for our model in order to show that RAG will be able to help us give more accurate models. Secondly, we utilise the keywords retrieved from the keyword detection models and retrieve it with the vector database. After we have retrieved the definitions with a high similarity score, we will then add the definitions together with the prompt itself before feeding the new concatenated query through the model to give the prediction. The flow of the experiments is shown below:

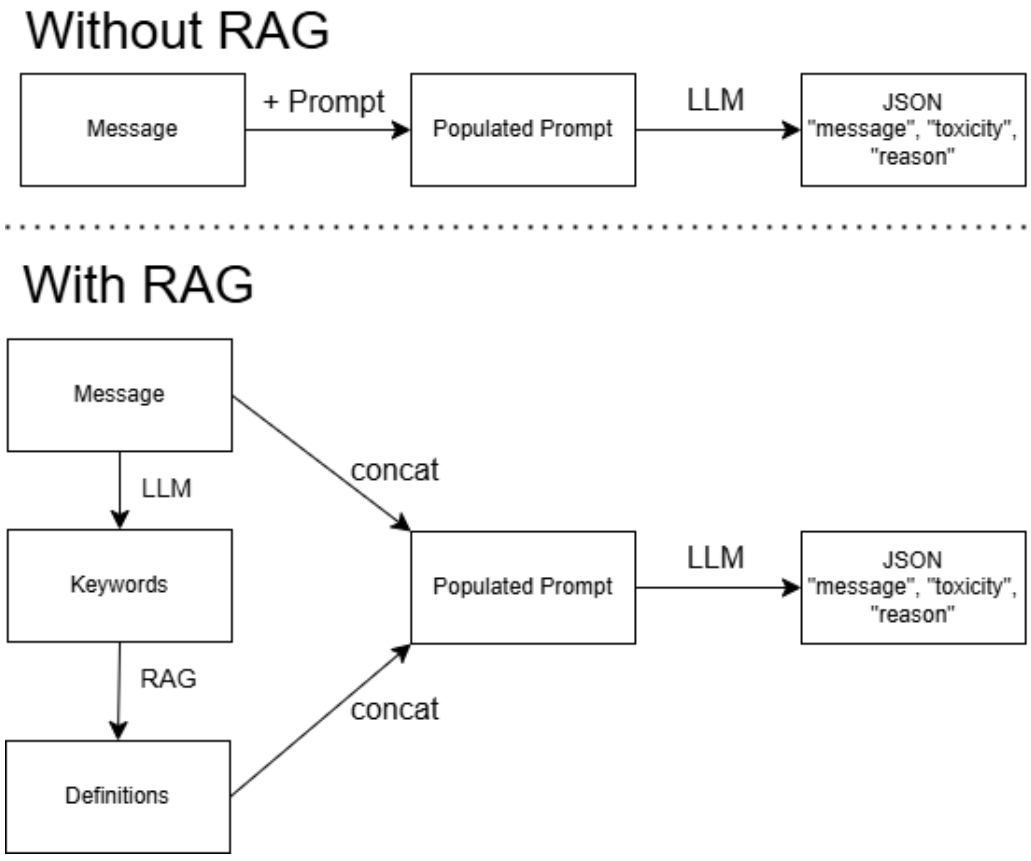

Figure 9: Difference in experiments as a basis for comparison

We then use the metrics that we will define soon in Section 5 to determine the prediction quality before RAG and after RAG.

5.6 Explainable AI

For our toxicity prediction prompt (Appendix A), we include a section that explicitly states the reason for flagging or banning a message as toxic. This aligns with the principles of Explainable AI (XAI), ensuring that users and moderators receive clear, interpretable justifications for each model decision.

By incorporating retrieved definitions of game-specific jargon and region-specific slang into the prompt, the model is better equipped to generate transparent explanations — for example, specifying whether the flagged term was identified as hate speech, a personal insult, or a psychological slur. This not only improves user trust in the system but also allows developers or moderators to audit model decisions more effectively.

Additionally, explainability is crucial in high-stakes environments such as online multiplayer games, where incorrect bans can damage user experience and community trust. Our design prioritizes interpretability without sacrificing model performance by leveraging prompt-based LLM reasoning alongside contextual grounding from the vector database.

6 Results

The results of the approach detailed in Section 4 are as follows. Firstly, let us define the metrics that we use to judge both our keyword detection and toxicity prediction tasks.

6.1 Metrics

6.1.1 Keyword Detection – Jaccard Similarity

We compare the extracted keywords to a set of manually extracted keywords and use **Jaccard Similarity** metric to evaluate how effective the LLM is at identifying keywords. The Jaccard Similarity is defined as follows:

Let K_{LLM} be the set of keywords extracted by the large language model and let K_{GT} be the set of manually extracted keywords (i.e., ground truth). Then the Jaccard Similarity is defined as:

$$\text{Jaccard}(K_{\text{LLM}}, K_{\text{GT}}) = \frac{|K_{\text{LLM}} \cap K_{\text{GT}}|}{|K_{\text{LLM}} \cup K_{\text{GT}}|}$$

Here, $|K_{\text{LLM}} \cap K_{\text{GT}}|$ represents the number of correctly predicted keywords that overlap with the ground truth, while $|K_{\text{LLM}} \cup K_{\text{GT}}|$ represents the total number of unique keywords across both sets.A higher Jaccard similarity indicates better agreement between the LLM's predictions and the ground truth, reflecting more accurate keyword extraction performance.

6.1.2 Toxicity Prediction – Precision, Recall & F1-Score

We use a few data science metrics to determine the efficacy of our models. We first define said metrics. The metrics are precision, recall and F1-score and are defined as such:

Let TP be the number of true positives, let FP be the number of false positives, and let FN be the number of false negatives. Then, the metrics are:

$$\text{Precision} = \frac{TP}{TP + FP}$$

where precision is the ratio of true positives to number of predicted positives, consisting of true positives and false positives.

$$\text{Recall} = \frac{TP}{TP + FN}$$

where recall is the ratio of true positives to number of actual positives, consisting of true positives and false negatives.

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Where the F1-score is the harmonic mean between precision and recall.

Specifically, given the nature of the problem, the F1-score would naturally be the best metric to optimise for as we are looking to both accurately ban toxic players. However, apart from that, we deem current systems to lack enough context to be able to catch toxic messages. Therefore, we prioritise the recall of our system. Precision is also considered, as there have been cases of gamers being falsefully banned and therefore will be looked at with our results.

6.2 Keyword Detection Results

We calculate the Jaccard Similarity by averaging the Jaccard Similarities across our 150 datapoints. From there, we can see that the Jaccard Similarity score achieved by our system is **0.687**, indicating a strong overlap between the predicted and actual keyword sets. In the context of text processing, this score reflects the model's effectiveness in capturing relevant and meaningful terms, with over two-thirds of the predicted keywords aligning with the ground truth. Such a level of similarity suggests that the system is consistently identifying core content features and demonstrates reliable semantic understanding. This result reinforces the model's capability in handling keyword detection tasks with a high degree of accuracy and relevance and therefore will ensure that our RAG process will be reliable enough such that it can query keywords to help improve the metrics that we use for toxicity prediction.

6.3 Toxicity Prediction Results

With our system, we see an increase in some of the metrics mentioned in 5.1.2.

Metric	Base Precision	RAG Precision	Base Recall	RAG Recall	Base F1-Score	RAG F1-Score	Support
False	0.96	0.98	0.88	0.87	0.92	0.92	122
True	0.62	0.62	0.86	0.93	0.72	0.74	28
Macro Average	0.79	0.80	0.87	0.90	0.82	0.83	150
Weighted Average	0.90	0.91	0.87	0.88	0.88	0.89	150

Table 1: Toxicity Prediction Results

We first focus on the F1-score. For the False class, the F1-score remains unchanged at 0.92, indicating that the model continues to handle negative samples with a high degree of reliability. Notably, the True class saw an increase in F1-score from 0.72 to 0.74, a meaningful improvement considering the inherent challenge of identifying minority or positive cases. This gain reflects better overall performance in detecting the True class.

Looking at the macro and weighted averages, we see further evidence of overall model improvement. The macro-average F1-score — which treats each class equally regardless of their frequency — increased slightly from 0.82 to 0.83. This rise reflects a more balanced performance across both the majority and minority classes, highlighting the model's improved ability to handle True cases without compromising performance on False cases.

Meanwhile, the weighted-average F1-score, which accounts for class imbalance by giving more weight to the majority class, also improved from 0.88 to 0.89. This indicates that the model's performance gains were not limited to niche cases but rather translated into a more robust and reliable classifier overall.

We then move on to recall, which is the second most important metric to improve after F1-Score. The model maintained a high recall for the False class (slightly decreasing from 0.88 to 0.87), but achieved a significant improvement for the True class, increasing from 0.86 to 0.93. This suggests that the updated model is now catching a greater number of true positives, which is especially valuable in our context of toxicity prediction.

Lastly, let us look at precision. The model improved in identifying False class instances correctly, increasing from 0.96 to 0.98, while True class precision remained stable at 0.62, indicating no drop in the accuracy of its positive predictions despite the increased recall.

In summary, the updated system demonstrates meaningful improvements in key performance metrics, particularly in handling the minority True class. The gains in recall and F1-score, combined with stable or improved precision, suggest that the model is now more effective at identifying true positives without compromising overall accuracy. The increase in both macro and weighted average F1-scores reinforces this trend, pointing to a more balanced and reliable model performance across classes.

7 Conclusion

This project proposed and evaluated a novel system for detecting game toxicity using LLMs enhanced by RAG mechanisms. Recognizing the limitations of existing models—especially in their failure to interpret game-specific or region-specific jargon accurately—we introduced a RAG-Keyword architecture that intelligently combines semantic relevance and character-level similarity via cosine similarity and normalized Levenshtein distance. This hybrid objective significantly improved the model's ability to interpret informal, typo-prone messages in high-speed gaming environments.

Through empirical evaluations on a multilingual, multi-regional dataset collected from real gameplay, the system demonstrated robust improvements in key performance metrics. The toxicity prediction component exhibited notable gains in recall and F1-score for minority classes without compromising the precision of its classifications. This reinforces the feasibility and practicality of deploying such a system in real-world gaming platforms where trust, fairness, and contextual nuance are essential.

By designing a pipeline that integrates LLMs, semantic search, and explainable AI, this project not only advances the state-of-the-art in game toxicity detection but also lays the groundwork for scalable, interpretable, and culturally aware moderation systems. The approach balances computational efficiency and model performance, making it viable for widespread deployment in modern multiplayer environments.

7.1 Code

The codebase attached to this project is organized into several directories, each serving a distinct purpose in the development and analysis of this project. The directories are as such:

1. **data/** houses the dataset files in JSON format, as mentioned in Section 5.2. These files contain the raw data used for training and evaluating the toxicity detection models.
2. **vector/** contains scripts related to vector operations, which consists of upsert.py to insert the definitions (Appendix A) from .csv file, query.py for querying the vector database and clear_namespaces.py to clear namespaces in pinecone.
3. **process/** contains scripts to process the JSON files into a .csv file, which is being used for our experiments.
4. **prompts/** contains prompt templates as well as utils.py, which are utility function that are used to format the prompt templates to feed into the models for prediction.
5. **inference/** contains scripts for running inference on the trained models. The folder contains both keyword.py and prediction.py for these models.
6. **experiments/** is dedicated to running and documenting various experiments related to the project. Base_predictions.py executes the "Without RAG" workflow in Figure 9, while complete_workflow.py executes the "With RAG" workflow in Figure 9. Lastly, we store all of our results in the results sub-folder.

The code is available at the following GitHub repository: https://github.com/kwangyy/game-toxicity

7.2 Further Improvements

For this project, further improvements can be made to the system itself. With new features in game, there will be new game-specific jargon that might come out to address these features. The new jargon has higher probabilities to be pointed out as a keyword. Therefore, when there are continuous words that have been flagged as a keyword, yet the score is low (which signifies that there is a lack of similarity with any words in the vector database, we can identify it as a new keyword for us to find the meaning of. We can then search up the keyword online with context to the game or to the country in social media websites such as Reddit or Twitter to find similar pieces of media. After that, we are able to extract the definition of the keyword through a LLM, allowing us to be up to date with new keywords whenever new updates occur with these games.

We are also able to extend this system to voice data, whereby we utilise a speech-to-text system in order for us to extract out the text. We then use the same system to identify keywords, as well as whether the person was being toxic. Due to the lack of data, we were not able to utilise this during this project.

We can also process our messages in batches instead of single messages. This is so that this provides better accuracy given the previous messages it had access to. However, due to the small sample size of the dataset, we could not implement batch processing for this project.

Lastly, with systems like these, there will be a high tendency for toxic players to continually obfuscate their messages to avoid detection. We can eventually feed the query into a LLM to rewrite it before we feed it through the rest of the pipeline in order to avoid this obfuscation. However, LLMs already do that quite well, where obfuscation techniques like leetspeak (e.g. H34LS PL34S3) and continuously adding spaces between words are processed behind-the-scenes in order to predict toxic messages accurately.

7.3 Acknowledgements

I would first like to acknowledge Prof Tong Xin for being my supervisor for this thesis. Without him, this thesis would not be possible. I would like to thank him for all the ideas that he has provided, as well as all improvements that he has suggested for this thesis.

Next, I would like to specially acknowledge my two friends, Mingqian Hu and Akmal Lawal, for playing games with me during this period. Without them, I would not be able to collect the data provided in this report. Data for this project was very hard to collect as it needed a lot of time, and I am grateful that they have played games with me so that we could collect real data.

I would also like to acknowledge a few datapoints provided by my other friends, namely Jet, Sebbie, and Elham. Sebbie and Elham are from Australia and therefore have introduced variety in my projects by providing me with data from the Oceania region for me to analyse.

I would like to thank those who have helped me throughout my journey during my time in NUS. I cannot express how much help I have received throughout my 3 years in NUS, and it has really propelled me to do even more in NUS. Thinking of graduating in 3 years would not be possible at all without help from people I've met, so to be even able to do that is not a solo effort at all.

Lastly, thank you to Nexon for being my entry into MMO gaming – without Maplestory, I would not have had a memorable childhood without Maplestory, and it really shaped my curiosity with different topics and has allowed me to sustain this curiosity even up till today.

Bibliography

Robertson, A. (2020, October 7). *Toxicity in gaming is dangerous. Here's how to stand up to it.* WIRED. https://www.wired.com/story/toxicity-in-gaming-is-dangerous-heres-how-to-stand-up-to-it/

Unbanster. (n.d.). *False positive bans – What they are and how to avoid them.* https://unbanster.com/false-positive-bans/

Blackburn, J., & Kwak, H. (2014). *STFU NOOB!: Predicting crowdsourced decisions on toxic behavior in online games*. Proceedings of the 23rd International Conference on World Wide Web (WWW '14 Companion), 793–798. https://doi.org/10.1145/2567948.2576937

Yang, Z. (2024). *Game On, Hate Off: A Study of Toxicity in Online Multiplayer Environments*. *Games Research and Practice*, 2(3), 45–60. https://doi.org/10.1145/3675805

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). *Attention is all you need*. Advances in Neural Information Processing Systems, 30.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Min, S., Yih, W.-T., & Riedel, S. (2020). *Retrieval-augmented generation for knowledge-intensive NLP tasks*. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in Neural Information Processing Systems* (Vol. 33, pp. 9459–9474). https://arxiv.org/abs/2005.11401Wolfe, C. (2023). *Decoder-Only Transformers: The Workhorse of Generative AI*. Substack. Retrieved from https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse

PromptHub. (2024). *Llama 3.1-8b*. Retrieved April 7, 2025, from https://www.prompthub.us/models/llama-3-1-8b

OpenAI. (2024). *API pricing*. Retrieved April 7, 2025, from https://openai.com/api/pricing/

Appendix A: Initial prompts

Keyword prompt (keyword.txt)

You are an expert keyword analyzer, with the task to analyze chat messages from online multiplayer games.

For each message, determine which keywords, if any, are present.

A keyword is a term that appears in a predefined whitelist.

Extract it from the message if it matches any of the following criteria:

- Common gaming phrases: "gg", "gl", "hf", "nt", "ez", "lol", "wp"

- Game-specific actions: "planting", "defusing", "save", "heal", "heals", "dps"

- Character or ability names: "spiderman", "psylocke", "hawkeye", "jeff"

- Internet slang and meme words: "goat", "lmao", "yawn", "noob"

- Words that are not commonly found in English: "chao", "kaopei", "xin", "bodoh", "nguyen", etc.

Rules:

- Strip punctuation and ignore case when matching keywords.

- Do not perform stemming or lemmatization (e.g., "heal" and "heals" are different).

- If no keywords are matched, return an empty list.

- Do NOT return code of any sort, and return JSON only. The JSON definition has been provided for you below.

Return a JSON consisting of the keys "message" and "keywords", where the "message" denotes the original message and the "keywords" are a list of keywords that have been predicted.

Examples:

{{

"message": "GG everyone!",

"keywords": ["gg"]

}}

{{

"message": "noob dps",

"keywords": ["noob", "dps"]

}}

{{

"message": "Can you heal me hawkeye?",

"keywords": ["heal", "hawkeye"]

}}

{{

"message": "xin chao"

"keywords": ["xin", "chao"]

}}

{{

"message": "stop pushing alone",

"keywords": []

}}

Toxicity Prediction Prompt (prediction.txt)

You are an expert in predicting toxicity.

Your task is to determine whether each message is toxic or non-toxic based on its content.

You are to label the message as **toxic** if it contains any of the following:

- Profanities that are aggressive to others (e.g., "fuck you", "stfu", "suck a dick")

- Hate speech or slurs (e.g., "china man", offensive references to race or religion)

- Personal insults (e.g., "noob", "low iq", "get a life")

- Sexual harassment or violent remarks (e.g., "did your mum get owned")

- Psychological attacks or ableist slurs (e.g., "brain disorder", "retard")

There are definitions that may or may not be provided for you.

If they are, use the keywords that are provided to you, and factor them into your decision in predicting toxicity.

Return a reason (or reasons) why they are predicted toxic or they are not predicted toxic.

Do NOT return code of any sort, and return JSON only. The JSON definition has been provided for you below.

If the message does **not** contain any of the above, mark it as **non-toxic**, even if it is sarcastic or expresses frustration.

Examples have been given to you:

{{

"message": "stfu stupid dog",

"toxicity": true,

"reason": "dog is being used as a derogatory term here, and stfu is aggressive to others."

}}

{{

"message": "GG WP everyone!",

"toxicity": false,

"reason": "this is a wholesome comment that tells everyone it was fun to play with them."

}}

{{

"message": "dps idiot can't even shoot",

"toxicity": true,

"reason": "this comment is toxic as the player is frustrated and is taking it out on the dps."

}}

{{

"message": "bro we lost again lol",

"toxicity": false

"reason": "this is a negative comment, however it is not toxic as it could be that the player is frustrated"

}}

Appendix B: Words and meaning, per namespace

Australia

Word	Meaning
cunt	can be affectionate or insulting depending on tone; common in aussie slang
fuck	the universal f-bomb; used for anger, emphasis, or insult
shit	general curse word; used to express frustration
bloody	mild swear word used for emphasis
bastard	insult implying someone is mean or cruel
dickhead	stupid or annoying person
wanker	jerk; often implies someone is self-centered or irritating
bugger	mild expletive; can also mean 'damn'
arsehole	jerk, rude person
dick	insult for a rude or stupid person
tosser	similar to wanker; an idiot
prick	insulting term for a man
drongo	someone who is dumb or useless
mong	offensive term for someone acting stupid
bitch	offensive when used toward women; also used to insult someone acting weak

Counter-Strike

Word	Meaning
planting	placing the bomb at the site
defusing	disarming the planted bomb
eco	saving money instead of buying weapons
buying	purchasing weapons, armor, or utility
saving	keeping your weapon for the next round
pushing	aggressively moving toward an area
rotating	moving to support another bomb site
entrying	being the first into a site
lurking	staying behind to catch rotating enemies
flanking	attacking from behind
boosting	lifting a teammate to a higher spot
peeking	quickly exposing yourself to check a spot
spraying	firing continuously
tapping	firing one bullet at a time

clearing	checking a location for enemies
shouldering	faking a peek to bait a shot
jump spotting	jumping to gain vision without being exposed
mollying	throwing a molotov
nading	using a grenade
smoking	deploying a smoke grenade
flashing	throwing a flashbang to blind enemies
wallbanging	shooting through walls
jiggling	quickly moving in and out of cover
refragging	trading a kill after a teammate dies
anchoring	staying on-site as the last line of defense

General

Word	Meaning
gg	good game; used after matches
gl	good luck
hf	have fun
nt	nice try
ez	easy win; often used mockingly
lol	laughing out loud
wp	well played
ff	forfeit; surrender vote
afk	away from keyboard
op	overpowered
nerf	reduce the power of something
buff	increase the power of something
carry	someone who wins the game for the team
feed	dying repeatedly to the enemy
tilt	getting mentally frustrated or emotional
camp	waiting in one place to ambush
bot	bad player or AI player
toxic	rude or unsportsmanlike behavior
main	your favorite or most played character
noscope	shooting without aiming down sights
headshot	direct hit to the head
frag	kill or eliminate an enemy
clutch	winning a tough situation alone
gank	surprise attack on an enemy (usually with teammates)

smurf	high-level player using low-ranked account
goat	greatest of all time
lmao	laughing my ass off
yawn	used to mock boredom
noob	beginner or bad player
sus	suspicious
cap	lie or false
flex	showing off
simp	overly submissive for attention (often romantic)
cringe	embarrassing or awkward
ratio	reply gets more likes than the original post
based	confident, unapologetic (non-conforming)
woke	socially aware (positive or sarcastic)
stan	obsessive fan
salty	bitter or upset
slay	doing something really well
lit	exciting or cool
dead	so funny you 'died' laughing
fire	awesome or amazing
clown	someone who made a fool of themselves
pog	expression of excitement
bet	agreement or confirmation ('okay', 'sure')
bruh	expression of disbelief or frustration
savage	brutally honest or bold
meme	a humorous image, video, or text that spreads online

League of Legends

Word	Meaning
ganking	ambushing a lane from the jungle
farming	killing minions for gold
roaming	leaving lane to assist other lanes
diving	attacking under enemy turret
flashing	using the flash summoner spell
ulting	using your ultimate ability
zoning	keeping enemies out of an area
peeling	protecting your carries by removing threats
engaging	starting a fight

disengaging	retreating from a fight
tanking	absorbing damage for your team
healing	restoring health
shielding	protecting teammates from damage
ksing	kill-stealing; taking a kill meant for someone else
splitpushing	pushing a side lane alone
camping	staying in one area to repeatedly gank
leashing	helping your jungler kill their first camp
warding	placing vision wards
sweeping	removing enemy wards
freezing	keeping the minion wave near your turret
shoving	quickly pushing the wave
backing	recalling to base
scaling	improving as the game progresses
invading	entering the enemy jungle early
snowballing	gaining momentum through early advantage

Marvel Rivals

Word	Meaning
adam	cosmic warrior with resurrection and shielding powers
blackpanther	stealthy melee assassin with burst and agility
blackwidow	tactical fighter using gadgets and stealth
capt	shield-wielding frontline with defensive buffs
cd	cloak and dagger duo combining stealth and burst
drstrange	dimensional mage with portals and shielding
groot	tanky support who heals allies and roots enemies
hawkeye	ranged marksman with precision arrows and traps
hela	necromancer who summons undead and controls space
hulk	durable tank with slam and leap-based melee control
torch	ranged fire caster with aerial mobility and AoE

invisible	invisible support who shields and vanishes allies
ironfist	melee bruiser with chi-empowered strikes and mobility
ironman	armored DPS with repulsors and flight
jeff	land shark with bite attacks and pressure mobility
loki	trickster using illusions, stealth, and disorientation
luna	ice mage with ranged control and team buffs
magik	teleporting swordfighter with portals and dark magic
magneto	controller manipulating metal and battlefield layout
mantis	support who stuns enemies and heals teammates
misterfantastic	stretchy control support who zones and entangles
moonknight	stance-switching melee bruiser with burst combos
namor	amphibious tank with water attacks and movement
peni	ranged support using a spider mech to zone enemies
psylocke	fast assassin with psychic melee attacks and mobility
punisher	mid-range tactical shooter with gadgets and grenades
thing	durable brawler with slam-based crowd control
rocket	explosive trap-setting ranged DPS with high burst
scarletwitch	chaos caster with hexes and unpredictable AoE
squirrel	agile disruptor who summons squirrels and flanks
spiderman	agile hero with web-swinging, control, and mobility
starlord	mobile ranged blaster with jukes and AoE tools
storm	aoe mage with wind knockbacks and lightning damage

thor	melee bruiser with hammer throws and lightning burst
venom	brawler with lifesteal and aggressive melee control
winter	tactical fighter with rifle, grenades, and mobility
wolverine	melee tank with self-heal, slashes, and pursuit
shooting	firing your weapon or basic attack
healing	restoring health to teammates
flanking	sneaking around to attack from the side
zoning	controlling an area with abilities
dodging	avoiding incoming damage or crowd control
blocking	reducing or negating damage taken
dashing	quick movement to reposition or escape
jumping	vertical movement to reposition
grappling	using mobility tools to pull or swing
ulting	activating your ultimate ability
scanning	checking for enemy positions
contesting	actively stopping the enemy from capturing
capturing	taking control of a point
defending	holding a position or payload
respawning	returning to the game after being eliminated
escorting	guarding the payload or objective
retreating	falling back to regroup or heal
engaging	starting a team fight
bursting	dealing large amounts of damage in a short time
dpsing	continuously dealing damage over time
tanking	soaking damage for your team
supportting	assisting allies with healing or buffs
shielding	absorbing damage with barriers
reviving	bringing a fallen teammate back
pinging	marking a location or enemy for teammates

Southeast Asia

Word	Meaning
puki	vulgar term for vagina

babi	pig; insult implying someone is dirty or lowly
lanjiao	penis (hokkien); very crude
lj	penis (hokkien); very crude
cheebai	vagina; vulgar term
jibai	vagina; vulgar term
cb	short for 'cheebai' (vagina); very vulgar
kanina	fuck your mother (hokkien); extremely offensive
knn	fuck your mother (hokkien); extremely offensive
kontol	penis; crude insult
anjing	dog; used to call someone despicable
ngentot	to fuck; harsh insult
jembut	pubic hair; vulgar term
putangina	your mother's a wh*re; extremely vulgar
gago	idiot or stupid person
leche	damn; mild curse expressing frustration
bobo	dumb or unintelligent
du ma	fuck your mother; very vulgar
cac	dick; extremely crude
nguyen	signifying a person from Vietnam; usually used as an insult
dit me	another form of fuck your mother
co cho	dog; derogatory term
djt me may	fuck your mom or you; vulgar insult
troi oi	oh my god
hee	vagina; extremely rude
hia	bastard or jerk
sat	animal; used as an insult
kwai	buffalo; insult for someone stupid
mae mueang	your mother; rude insult
pauk	penis
mee thu	bastard
myin	horse; used to call someone stupid
kyet	chicken; mocking term
ma le	slut

Teamfight Tactics

Word	Meaning
rolling	spending gold to refresh the shop
econing	saving gold to build interest
leveling	spending XP to reach a higher level
rerolling	low-level rolling for 1–2 cost units
pivoting	changing your comp mid-game
slamming	combining items early without waiting
scouting	checking enemy boards
positioning	arranging units on the board
stacking	putting items on a single carry
sacking	intentionally losing rounds to build economy
streaking	win or loss streak for extra gold
spiking	hitting a power spike from upgrades
contesting	competing with someone else for the same comp
rolling down	spending all your gold to hit upgrades
open forting	losing intentionally in early game
selling	removing units to get gold
griefing	denying units from others intentionally
healing	restoring team HP via augments or traits
mana burning	reducing enemy ability usage
tanking	placing units to absorb damage
clumping	grouping units close together
cornering	putting your carry in a corner for protection
donkey rolling	rolling without planning
cheesing	using a non-meta or off-beat strategy to win
slow rolling	saving gold and rolling above 50g for upgrades