COMP4801 Final Year Project

Game Testing and Evaluation Platform with

Machine Learning for Game Developers

# CritiQ

Final Report

Supervisor: Dr. Chim T. W.

Group Members:

Lee Chi Ho 3035785520 (**Writer**)

Cheng Pak Yim 3035784942

Siu Yuk Shing 3035779430

Date of Submission: 24/4/2024

## Abstract

The rapid development in Machine Learning (ML) and Natural Language Processing (NLP) has demonstrated revolutionary improvements in textual understanding and processing. However, such technologies have not been widely adopted for game review analytics on major review platforms, requiring manual data processing to extract valuable information. This project aims to investigate the potential of automated review analysis by harnessing the power of Natural Language Processing.

This project developed a full-stack web application using Spring Boot and React frameworks that incorporates NLP features, including Sentiment Analysis, Keyword Extraction, Topic Modeling, and Summarization. Results have shown high accuracy, efficiency, and performance in analyzing game reviews and could be easily adopted and integrated into many game review platforms. The project underscores the transformative impact that Large Language Models (LLMs) and NLP technologies could have on the gaming industry, particularly in the realm of game review analytics. It stands as a testament to the benefits of integrating ML and NLP into existing systems, paving the way for more efficient and informed game development processes.

## Note on Content

A comprehensive analysis of the Natural Language Processing (NLP) components is available in Cheng Pak Yim's Final Report, and an in-depth examination of the Frontend development is presented in Siu Yuk Shing's Final Report. This document will provide a concise overview of their contributions. The majority of the foundational work was completed by myself during the first semester and my contribution during the second semester involved the integration of their respective parts into the system and the enhancement of the backend infrastructure.

## Acknowledgment

We would like to extend our profound gratitude and heartfelt appreciation to our project supervisor, Dr. Chim T.W. His invaluable guidance, unwavering support, and insightful mentorship have been instrumental in the successful completion of this project. We are truly grateful for his time, effort, and the faith he has shown in us since the start of the project. His mentorship has been a vital component of our project's success, and for that, we express our deepest appreciation.

# Table of Contents

*Abstract* ..... *ii*

*Acknowledgment* ..... *iii*

*List of Figures* ..... *vi*

*List of Tables* ..... *x*

*1. Introduction* ..... *1*

1.1. Background ..... 1

1.2. Objectives ..... 1

1.3. Deliverables ..... 2

1.4. Outline ..... 2

*2. Related Work* ..... *3*

*3. Methodology* ..... *6*

3.1 Machine Learning and Natural Language Processing ..... 6

3.1.1 Sentiment Analysis ..... 6

3.1.2 Topic Modeling ..... 20

3.1.3 Keyword Extraction ..... 23

3.2 Frontend Web Application ..... 26

3.2.1 Technologies Involved ..... 26

3.2.2 Design Approach ..... 26

3.2.3 Frontend Authentication ..... 27

3.3 Backend Technologies ..... 29

3.3.1 Spring Boot Server Application ..... 29

3.3.2 Authentication with JWT ..... 30

3.3.3 Email Service ..... 31

3.3.4 Database ..... 33

3.3.5 Object Storage ..... 34

3.3.6 Continuous Integration/Continuous Delivery (CI/CD) ..... 35

3.3.7 Message Queue ..... 37

3.3.8 Hosting ..... 39

3.3.9 Monitoring ..... 40

*4 Results* ..... *41*

4.1 Sentiment Analysis ..... 41

4.2 Topic Modeling and Keyword Extraction ..... 49

4.3 Game Aggregated Review ..... 64

4.4 Game Analytics ..... 67

4.5 Web Application ..... 70

4.5.1 Toolbar ..... 70

4.5.2 Login and Registration ..... 71

4.5.3 Forgot Password Page and Reset Password Page ..... 74

4.5.4 Landing Page ..... 77

4.5.5 Profile Page ..... 81

4.5.6 Search Result Page ..... 85

4.5.7 Game Page ..... 89

4.5.8 Game Analytic Page ..... 94

4.5.9 Review Page ..... 98

**4.5.10 Progressive Web Application** ..... **101**
**4.6 Web Scraping** ..... **104**
**4.7 Backend System** ..... **105**
**4.4.1 CI/CD** ..... 106
**4.4.2 Message Queue Implementation** ..... 107
**4.4.3 API Endpoints and Database** ..... 109
**4.4.4 Authentication** ..... 112
**4.4.5 API Security** ..... 114
**4.4.6 S3 Bucket Storage** ..... 115
**4.4.7 Stability and Testing** ..... 117
***5 Difficulties Encountered*** ..... ***119***
**5.1 HTTPS and SSL Certificate** ..... **119**
**5.2 Insufficient Computational Capacity** ..... **121**
**5.3 Message Queue Disconnection** ..... **123**
***6 Limitations*** ..... ***125***
**6.1 Noise in Review Data** ..... **125**
**6.2 LLM Hallucination** ..... **126**
***7 Future Works*** ..... ***129***
**7.1 Integration with Business Intelligence Tools** ..... **129**
**7.2 LLM for 3-Way Sentiment Analysis** ..... **130**
**7.3 Topic Modeling Spam Filtering** ..... **132**
**7.4 Game Aggregated Review Automation** ..... **133**
***8 Schedule*** ..... ***135***
***9 Work Distribution*** ..... ***135***
***10 Conclusion*** ..... ***136***

## List of Figures

Figure 1: Usual stages in Sentiment Classification. (a): Six usual stages in Sentiment Classification. (b): Overall framework of section Sentiment Analysis of the project. Two additional stages were added and labeled in pink. ..... 7
Figure 2: Steam automated comment filtering. The sensitive word was replaced by consecutive heart symbols. ..... 8
Figure 3: Number of reviews in each sentiment class in the cleaned dataset with a 4.89: 1 positive to negative ratio. ..... 10
Figure 4:Procedure of further data cleaning on the cleaned dataset. ..... 10
Figure 5: Number of appearances of top 20 frequent words in the cleaned dataset after further data cleaning. (a): Words in the cleaned dataset with both positive and negative reviews. (b): Words in the cleaned dataset with only positive reviews. (c): Words in the cleaned dataset with only negative reviews. (d): Common words in (b) and (c). ..... 10
Figure 6: Model structure of CNN in model GloVe-CNN. ..... 15
Figure 7: Customized further data cleaning to each model. Left: TFIDF-RF. Middle: GloVe-CNN. Right: BERT ..... 17
Figure 8: Example output of BERTopic using KeyBERT and Llama2 to name the topics. ... 24
Figure 9: Two game reviews and the response from ChatGPT hosted by Azure when prompting to classify their sentiment. ..... 24
Figure 10 JWT Claims extracted from the Access Token user received on login ..... 31
Figure 11 Function and SQL that remove stale and outdated authentication and refresh token ..... 31
Figure 12 Database Entity Relations Diagram ..... 33
Figure 13 Sample Code to upload a file to the S3 Bucket ..... 34
Figure 14 Digital Ocean Spaces Dashboard ..... 35
Figure 15 Jenkinsfile pipeline written for deploying the Backend Server and NLP Server with the use of docker and dockerfiles ..... 36
Figure 16 Message Queue Structure for Supporting Inter-process Machine Learning Application ..... 38
Figure 17 RabbitMQ Management Panel showing all the queue connections. ..... 38
Figure 18 Grafana Dashboard displaying uptime, CPU, and Memory Utilization by the Spring Boot Application. ..... 40
Figure 19 Grafana Dashboard shows the database connection pool size, maintaining a stable connection to the database. ..... 40
Figure 20: Results of all models trained with 120K balanced dataset. (a): Weighted Average F1-score on all models. (b): Weighted Average Precision on all models. (c): Weighted Average Recall on all models. ..... 41
Figure 21: Precision of both sentiments on balanced validation set by models trained with 120K imbalanced and balanced datasets. (a): Positive. (b): Negative. ..... 42
Figure 22: Recall of both sentiments on balanced validation set by models trained with 120K imbalanced and balanced datasets. (a): Positive. (b): Negative. ..... 43
Figure 23: Results of all models trained with 240K balanced dataset. (a): Weighted Average F1-score on all models. (b): Weighted Average Precision on all models. (c): Weighted Average Recall on all models. ..... 43
Figure 24: Results of all models trained with 480K balanced dataset. (a): Weighted Average F1-score on all models. (b): Weighted Average Precision on all models. (c): Weighted Average Recall on all models. ..... 44

Figure 25: Results of all models trained with balanced datasets of all three sizes. (a): Weighted Average F1-score on all models. (b): Weighted Average Precision on all models. (c): Weighted Average Recall on all models ..... 44
Figure 26: Results of all models trained with imbalanced datasets of all three sizes. (a): Weighted Average F1-score on all models. (b): Weighted Average Precision on all models. (c): Weighted Average Recall on all models ..... 45
Figure 27: ROC-AUC of all models trained with the balanced dataset. (a): on the imbalanced validation set. (b): on the balanced validation set. ..... 47
Figure 28: ROC curve of BERT fine-tuned on 240K balanced training set. (a): on the imbalanced validation set. (b): on the balanced validation set. ..... 47
Figure 29: Median inference time of original and ONNX model on different machines. (a): Machine 1 (Windows i5-8250U). (b): Machine 2 (Apple M1 Max) ..... 48
Figure 30 Technologies for Keyword Extraction and Topic Modeling on Review Creation ..... 49
Figure 31 Flowchart for Keyword Extraction, Topic Modeling, and Summarization ..... 50
Figure 32 Topic Coherence Score example of a Topic consisting of 3 words against a Reference Corpus. https://towardsdatascience.com/understanding-topic-coherence-measures-4aa41339634c ..... 51
Figure 33 C_NPMI Measurement of 3 Topic Modeling Models and their performance ..... 52
Figure 34 Inverted RBO Measurement of 3 Topic Modeling Models and their performance 53
Figure 35 pyLDAvis web UI showing the most common topic its top keywords and the distance map ..... 54
Figure 36 BERTopic's provided visualization interface similar to LDAvis, displaying topic overlap and size ..... 54
Figure 37 Cross-model comparison on a similar topic (crashes and bugs) ..... 55
Figure 38 Cros- model comparison on a similar topic (horror puzzle game) ..... 55
Figure 39 Topics for Action Games generated by Trained Topic Modeling model ..... 56
Figure 40 Transferred Topics for Action Games generated by Trained Topic Modeling model and LLM ..... 56
Figure 41 (a) A warning sign in review cards shown in-game and on all reviews pages; (b) a "Possible Spam" phrase added in the review analysis section ..... 59
Figure 42 Sentiment generated per aspect for review ..... 62
Figure 43 Web Application Displaying Aspects sorted by their sentiments ..... 62
Figure 44 Keywords Section only display found aspects ..... 62
Figure 45 Web Application Displaying Summary of the Review ..... 63
Figure 46 Flowchart for Game Aggregated Review Generation ..... 64
Figure 47 Display of the Aggregated Review Summary in the Web Application's Game Page ..... 65
Figure 48 JSON result from gameAnalytic API ..... 65
Figure 49 Web application's toolbar design. (a): Toolbar design for desktop viewport. (b): Toolbar design for the mobile viewport. (c): Avatar icon button drop-down menu ..... 70
Figure 50 Web application's register modal popup ..... 71
Figure 51 Register modal input validations ..... 72
Figure 52 Web application's login modal popup ..... 72
Figure 53 Web application's forget password page design. ..... 74
Figure 54 Reset password email ..... 74
Figure 55 Web application's reset password page design ..... 75
Figure 56 Web application's reset password page with invalid token ..... 76
Figure 57 Landing Page of the Web Application, showing the logo, short descriptions along with 5 carousels showing popular titles ..... 77

Figure 58 Game Card within Landing Page carousel, showing the game "Starfield", along with its score, number of reviews, favorites and wish lists ..... 79
Figure 59 Game Card showing the Early Access game "Palworld", with a purple box displaying "Early Access" ..... 79
Figure 60 Carousel with games card and arrow control to navigate the carousel horizontally ..... 80
Figure 61 Landing Page in a Mobile Viewport with navigation buttons disabled and controls replaced by dragging gestures ..... 80
Figure 62 Profile Page of a Logged In User, showing the ability to toggle the privacy of their account at the top right ..... 81
Figure 63 Profile Page of a Logged In User in a mobile viewport, where elements are stacked vertically ..... 82
Figure 64 Profile page of a Private User account with all information except name hidden ..... 83
Figure 65 3-Dots Button revealing the username and profile banner update button to the account owner ..... 83
Figure 66 Profile Banner update UI, allowing user to preview prior to confirmation ..... 83
Figure 67 Verified Account with a Green Checkmark that displays "Verified User" on hover ..... 84
Figure 68 Default sorting order of "Latest" can be changed by clicking the button, revealing two other options ..... 84
Figure 69 Web application’s search result page design. (a): Search result page design for desktop viewport. (b): Search result page design for mobile viewport. ..... 85
Figure 70 Search result page searching example with game name consisting “Cyber” returning 2 results ..... 85
Figure 71 Advanced search modal for search result page ..... 86
Figure 72 Web application's game page design ..... 89
Figure 73 Game detailed information popup modal ..... 90
Figure 74 Favorite and Wishlist Buttons will turn red if user has already favorited and wish listed this game ..... 91
Figure 75 Snack bar display for unauthenticated user trying to favorite or wish list a game ..... 91
Figure 76 User with existing review will not be able to create new review and can view their current review and have the ability to edit their review ..... 92
Figure 77 Review Section change to edit review when user click on the "Edit Review Button" ..... 93
Figure 78 Review page with "Edited At" time showing the time of review being edited ..... 93
Figure 79 Edit Review button hidden with text displaying Time until next available edit. ..... 93
Figure 80 Game Statistics and Review Statistics of the Game Starfield in the Analytics Page ..... 95
Figure 81 Player Statistics and Wishlist & Favorite Statistics of the Game Starfield in the Analytics Page ..... 96
Figure 82 Floating Action Button that tracks the user's current section based on the viewport and allows for quick navigation ..... 97
Figure 83 Web application's review page design ..... 98
Figure 84 Review Section that shows the NLP solutions, including Sentiment Analysis, Main Topic, Keywords, and Short Summary ..... 100
Figure 85 PWA install button on the Chromium-based Microsoft Edge browser ..... 102
Figure 86 CritiQ PWA searchable in MacOS Spotlight Search ..... 102
Figure 87 Sample Database Records of Scraped Games from Steam ..... 104
Figure 88 Backend Solution Architecture Graph ..... 105
Figure 89 Jenkins deployment User Interface with different stages of deployments. ..... 106

Figure 90 RabbitMQ user interface showing all of the created queues ..... 107
Figure 91 sample JSON output for LLM token usage in message queue output response ... 108
Figure 92 API call to /findGameById to fetch specific game information finishes in 65ms 109
Figure 93 API call to /findGamesWithSearch to perform exhaustive game search finishes in 251ms ..... 110
Figure 94 Current Database Plan with 2GB RAM offers a maximum of 150 concurrent connections ..... 111
Figure 95 Spring Boot utilizes 80 concurrent connections to the database ..... 111
Figure 96 Access Control by verifying the user based on the JWT sent in HTTP requests using the @AuthenticationPrincipal annotation. ..... 114
Figure 97 Access Control by verifying the user's role based on JWT sent in HTTP requests before method invocation using the @PreAuthorize annotation. ..... 114
Figure 98 403 Forbidden Error on Unauthorized Access to Protected API endpoints ..... 115
Figure 99 Fetching of Cached Image(s) can be performed within 50ms ..... 115
Figure 100 Uploader Header is set to the user's name during file upload ..... 116
Figure 101 Backend Load-Testing using Gatling, showing the ability to sustain 60 active users performing complex queries. ..... 118
Figure 102 Google Chrome Browser Error Page on Visiting Website with a self-signed digital certificate ..... 120
Figure 103 Certificate Viewer on Backend Domain Address using the Google Chrome Browser ..... 121
Figure 104 10% CPU and 25% Memory usage for Machine Learning Models in new Virtual Machine ..... 122
Figure 105: Code snippet to initiate the RabbitMQ connect with heartbeat check disabled. 124
Figure 106 Example of LLM hallucination, generating keywords that do not exist in the review ..... 127

## List of Tables

Table 1 Distribution of reviews based on genres/description out of the top 10 genre/description. ..... 21
Table 2: Percentage change of all models trained with balanced datasets with different sizes. Up: Weighted Average F1-score. Bottom left: Weighted Average Precision. Bottom right: Weighted Average Recall. ..... 44
Table 3: Percentage change of all models trained with imbalanced datasets with different sizes. Up: Weighted Average F1-score. Bottom left: Weighted Average Precision. Bottom right: Weighted Average Recall ..... 45
Table 4: Weighted Average F1-Score of all models on the imbalanced validation sets. ..... 46
Table 5: Weighted Average F1-score of all models on the balanced validation sets. ..... 46
Table 6: Median Speedup of inference time of both machines. ..... 48
Table 7 Proposed Schedule for the project ..... 135
Table 8 Work Distribution Table of the project ..... 135

## Abbreviations

| Abbreviations and Acronyms | Full Term / Definition |
|---|---|
| ACID | Atomicity, Consistency, Isolation, Durability |
| API | Application Programming Interface |
| BERT | Bidirectional Encoder Representations from Transformers |
| BI | Business Intelligence |
| BoW | Bag-of-Words |
| CA | Certificate Authority |
| CDN | Content Delivery Network |
| CI/CD | Continuous Integration/Continuous Delivery |
| CNN | Convolutional Neural Network |
| CRUD | Create, Read, Update, Delete |
| CTM | Contextualized Topic Model |
| EDA | Exploratory Data Analysis |
| GloVe | Global Vectors for Word Representation |
| HTTP | Hypertext Transfer Protocol |
| LDA | Latent Dirichlet Allocation |
| LLM | Large Language Model |
| ML | Machine Learning |
| MUI | Material UI |
| NLP | Natural Language Processing |
| RAG | Retrieval-Augmented Generation |
| RTT | Round-Trip-Time |
| RWD | Responsive Web Design |
| SBERT | Sentence BERT |
| SDK | Software Development Kit |
| SSL | Secure Socket Layer |
| TF-IDF | Term Frequency-Inverse Document Frequency |
| TLDR | Too Long; Didn’t Read |
| UI | User Interface |
| UI/UX | User Interface/User Experience |
| URL | Uniform Resource Locator |
| VPS | Virtual Private Server |

# 1. Introduction

This section discusses the Project Background (Section 1.1), Objectives (Section 1.2), Deliverables (Section 1.3), and Outline (Section 1.4).

## 1.1. Background

Presently, game review platforms such as Steam and OpenCritic are predominantly designed to serve the needs of end-users. The functionalities offered by these platforms, particularly in terms of review analysis and filtering, are often limited. Consequently, this poses a challenge for developers seeking to derive meaningful insights from user opinions.

This project centers around the creation of a comprehensive full-stack review website with a primary objective of leveraging machine learning (ML) and natural language processing (NLP) techniques to empower game developers by eliminating the need for labor-intensive manual analysis and allowing game developers to make data-driven decisions. The project aims to revolutionize the review process by implementing advanced data processing and visualization capabilities, automating the analysis of reviews, and generating insightful feedback (Lin et al., 2019b) for developers. By harnessing the power of NLP, the website seeks to extract valuable information from user reviews, identify patterns and trends, and provide developers with detailed feedback that can improve their software development practices through easy-to-understand visualization. Developers can gain deeper insights and actionable intelligence from the reviews, enabling them to enhance their game and optimize the overall experience. This project represents a significant step towards empowering game developers through cutting-edge technologies and facilitating continuous improvement in their game development processes through automation of review analysis and data collection and aggregation.

## 1.2. Objectives

The five main objectives of this project are listed below.

- Research into NLP, including tokenization, stemming, and stopwords removal.
- Provide feedback using Sentiment Analysis, Topic Modeling, and Keyword Extraction.
- Perform web-scraping for data extraction, processing, and model training.
- Develop a scalable full-stack modern web application.
- Provide intuitive data visualization to users in the web application.

## 1.3. Deliverables

This project aims to deliver a full-stack cloud-native web application, including a deployable frontend webpage, scalable backend system, and database and distributed ML models that support the NLP functionalities of the application.

## 1.4. Outline

This report will present a comprehensive analysis of the project, focusing on key sections, including Related Work (Section 2), Detailed Methodologies (Section 3), Current-stage Results (Section 4), Difficulties encountered (Section 5), Limitations (Section 6), Future Works (Section 7), Schedule (Section 8), Work Distribution (Section 9) and a Conclusive summary (Section 10).

## 2. Related Work

Game reviews are user-generated posts that provide feedback, opinions, and discussions about specific games. They focus on evaluating the gameplay, graphics, and overall experience of the game. To better understand the characteristics of game reviews, related literatures were reviewed. Lin et al (2019b) conducted a thorough empirical study of game reviews to analyze their length, topics and relationship with players' playtime when writing the reviews. Comparisons were made between positive and negative reviews, early-access reviews and non-early-access reviews, indie games reviews and triple-A games reviews. Guzsvinecz & Szűcs (2023) analyzed the length and distribution of sentiments in over 35 million game reviews from 11 popular genres, such as Action, Racing and Sports.

However, special characteristics of the text in game reviews pose significant difficulty in the task of sentiment classification, in which the performance of the classifiers may be diminished. Viggiato et al (2022) identified six major characteristics. First, game reviews often come with frequent uses of contrast conjunctions as both advantages and disadvantages of the game are pointed out. Second, game reviews include words that is usually viewed negatively in other context, yet in a neutral or positive manner in game reviews, such as, "kill", "zombie", and "fire" in First Person Shooters and Action games. Third, sarcasm was frequently observed in game reviews. It occurs when a positive text was used to convey a negative attitude, or vice versa. The remaining three characteristics are unclearness, game comparison with another game or with a previous version of itself, and mismatched recommendation.

Sentiment analysis is one of the numerous aspects of NLP that aims to extract sentiments and opinions from texts (Birjali et al., 2021). It has been well applied in various domains, such as analyzing customers' product reviews, establishing a reliable recommendation system based on reviews, and public healthcare monitoring (Birjali et al., 2021). As for the gaming industry, various researchers attempted to classify the sentiment of comments using various machine learning models and deep neural networks.

Tam et al (2021) conducted a comparative study about the ability of machine learning models to classify around 15K game comments and reviews scrapped from Steam and Metacritic. Three-class sentiment labels, i.e., Positive, Negative, and Neutral, were first created with pre-trained sentiment analysis models, such as VADER, on reviews. The Synthetic Minority Oversampling Technique (SMOTE) was then applied to create a balanced dataset, tackling data

imbalance in neutral and negative reviews over positive reviews. Five machine learning algorithms, which are Logistic Regression (LR), Multinomial Naïve Bayes (MNB), Support Vector Machine (SVM), Multi-layer Perceptron Classifier (MLP), and Extreme Gradient Boosting Classifier (XGBoost) were then applied to build sentiment classifiers with both imbalanced dataset and balanced dataset. Results suggested that training with a balanced dataset with oversampling significantly improved the performance of the majority of models.

Ruseti et al (2020) conducted a study about three-class sentiment classification on a 117K dataset with game reviews from Metacritic. Reviews were first classified into positive, neutral, or negative based on the score on Metacritic. Then a range of ML models, which were SVM, MNB, and deep neural networks (DNN), combined with bag-of-words (BoW), word2vec embedding, or Universal Sentence Encoder, were trained on a balanced dataset to classify the reviews. All models achieved accuracy between 61% and 67%.

Al Mursyidy Fadhlurrahman et al (2023) applied Bidirectional Encoder Representations from Transformers (BERT), Bi-directional Long-short Term Memory (BiLSTM), and Bi-directional Gated Recurrent Unit (BiGRU) on two class sentiment classification with a balanced dataset containing 7K comments from 10 most reviewed games on Steam. They then presented another model, BERT-BiLSTM-CRF, to enhance sentiment classification over fine-tuned BERT on the dataset. The original fine-tuned BERT model achieved 0.88 in F1 score, accuracy, and recall.

The above-mentioned related works provide the expected sentiment classification rate in the context of game comments and reviews. However, works that compare the classification performance of game comments with different architectures on a large training and testing dataset are rare. As a result, our research in sentiment classification contributes to the aforementioned works as an experimental expansion by comparing the real-life performance of various approaches of feature extraction and model architectures.

Topic modeling is one of the numerous aspects of NLP that compresses a set of documents and returns a set of highly representative topics that describe the content accurately and coherently (Churchill & Singh, 2022). Due to the nature of the length of game reviews, and the rareness of applying topic modeling on game reviews, literature reviews regarding short text topic modeling on game reviews and social media posts were conducted.

Yu et al (2022) applied Latent Dirichlet Allocation (LDA) to explore the prominent topics in reviews from Dark Soul 3 and Dark Soul 1 separately, which the former is a sequel of the latter, and compare the common topics within the two games. 14 and 15 topics were uncovered respectively from a total of 130K English reviews from Steam. Topics uncovered reflected players enjoyed the combat, character, overall experience, and difficulty of both games and other commonly found aspects, such as graphics and gameplay.

A similar approach was also taken by (Stepien, 2021) to analyze the topic in 3 popular games: DOTA2, PUBG, and GTA5. LDA and LDA Sequential were applied to uncover the distribution of changes of shares of topics by training both models on reviews scrapped from Steam of each game. Results suggested that changes in shares of topics were observed when major game updates were released, or major tournaments were organized.

More advanced topic Modeling models have been applied to analyze different short texts in social media posts. Eagger & Yu (2022) compared the ability of four common topic modeling techniques, LDA, Non-negative Matrix Factorization (NMF), Top2Vec, and BERTopic, in analyzing the topics in Twitter posts regarding travel and the COVID-19 pandemic. 31800 unique tweets were collected from Twitter, and comparisons were drawn between the created topics and their keywords of LDA and NMF, and that of Top2Vec and BERTopic. Results suggested BERTopic and NMF were effective in analyzing Twitter data.

A similar comparison of topic models was conducted by Gan et al (2024) to compare three topic modeling methods: LDA, Top2Vec and BERTopic, in analyzing the latent topic in Twitter and Weibo posts regarding the topic ChatGPT. Result suggested generated topics from BERTopic were better segmented, more independent, with clear semantics understanding in both English and Chinese(Jaiswal et al., 2023).

The above-mentioned works provides a baseline regarding the ability of various topic Modeling techniques. Yet, works related to topic modeling on game reviews only focused on training separate models on each game. Works that compare the performance of applying a single topic modeling model across multiple games were hardly found. As a result, our research in topic Modeling contributes to the works as an experimental approach of applying a single trained topic model to analyze various common aspects of games in different games.

# 3. Methodology

The project can be divided into three main sections which are Machine Learning and Natural Language Processing (Section 3.1), Frontend Web Applications (Section 3.2), and Backend Technologies (Section 3.3). The methodology of development will be discussed in detail.

## 3.1 Machine Learning and Natural Language Processing

This section discusses three NLP tasks which are Sentiment Analysis (Section 3.1.1), Topic Modeling (Section 3.1.2), and Keyword Extraction (Section 3.1.3).

### 3.1.1 Sentiment Analysis

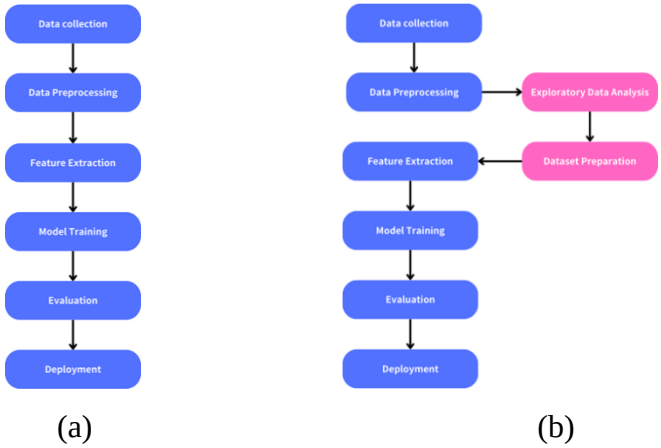
This section presents the Problem Definition (Section 3.1.1.1), Data Preparation (Section 3.1.1.2), Text Preprocessing (Section 3.1.1.3), Exploratory Data Analysis (EDA) (Section 3.1.1.4), Dataset Preparation (Section 3.1.1.5), Feature Extraction and Model Selection (Section 3.1.1.6), Model Implementation (Section 3.1.1.7), Model Training (Section 3.1.1.8), Model Evaluation (Section 3.1.1.9), and Model Deployment (Section 3.1.1.10).

#### 3.1.1.1 Problem Definition

Sentiment analysis is a crucial task for both game developers and potential players. It enables the former to understand the feedback and preferences of the gaming community, and the latter to form a clear and unbiased impression of the game's expected experience. This can facilitate better decision-making for both parties, such as improving game quality, features, and bug fixes, and making informed purchase choices. However, manually reading and analyzing all the comments and reviews about a game is impractical and inefficient, especially for popular titles that generate a large volume of text data. Therefore, automated sentiment analysis can provide a useful and convenient way to obtain a summary of the community's opinions and sentiments toward a game.

The process of sentiment analysis typically consists of six stages (See Figure 1 (a)). First, text data is collected from relevant sources or platforms, and structured and stored in a suitable format, forming a dataset. Second, data preprocessing is applied to the text dataset to remove noise, and irrelevant information, and reduce data dimensionality. Third, feature extraction is performed on the preprocessed text data to create a feature space that can be used by machine learning or deep learning models. Fourth, a model is implemented and trained on the extracted features to learn how to classify the sentiment

polarity of the text data, such as positive, neutral, or negative. Fifth, evaluation is conducted on separate testing or validation datasets to measure the performance of the trained model and select the best one. Sixth, the selected model is deployed to real-world scenarios on a system.




(a) (b)

*Figure 1: Usual stages in Sentiment Classification. (a): Six usual stages in Sentiment Classification. (b): Overall framework of section Sentiment Analysis of the project. Two additional stages were added and labeled in pink.*

As mentioned in Section 2, Related Works, it is not uncommon to find a huge imbalance in several positive and negative reviews in a game. Also, research from 2015 showed that the performance of ML models is positively correlated with the size of the training dataset in sentiment classification (Prusa et al., 2015). Therefore, three research questions were created to guide the process of selecting the best performant model in sentiment classification of game reviews. These questions are in the following.

RQ1: Does an imbalance training dataset hamper model performance?

RQ2: What is the relationship between dataset size and performance?

RQ3: What is the best model with little hyperparameter selection?

Due to the research questions, compared to the aforementioned common stages of sentiment classification, two stages, EDA and Dataset Preparations, were performed after Data Pre-processing and before Feature Extraction to understand the distribution of data and prepare corresponding datasets for RQ1 and RQ2. Hence, the framework of this section involves eight stages (See Figure 1 (b)).

#### 3.1.1.2 Data Preparation

An existing dataset created by Sobkowicz (2017) with reviews scrapped from Steam was selected. The dataset contains over 6.4 million publicly available English reviews from different games and genres on Steam. The dataset contains five columns, which are: [‘app_id’, ‘app_name’, ‘review_text’, ‘review_score’, ‘review_votes’], with each representing the id of the game on Steam, the name of the game, the review text, an

indicator whether the review recommends the game, and an indicator whether the review was recommended by another user respectively. The sentiment of each comment was labeled by the column ‘review_score’, whether a ‘1’ indicates a positive review, and a ‘-1’ indicates a negative review, as there are two distinct values in the column ‘review_score’, which was either ‘1’ or ‘-1’. All ‘-1’ labelings in column ‘review_score’ were converted to ‘0’ for the convenience of model training, as typically, labels for classes began from ‘0’.

Selecting an existing dataset with a large number of comments saves a significant amount of time in data collection from creating a scraping program and running the data scraping program, speeding up the development process.

#### 3.1.1.3 Text Preprocessing

Data cleaning is first performed on the cleaned dataset. First, rows with empty values in columns ‘app_name’ or ‘review_text’ were removed. Then, unhelpful comments containing merely the phrase ‘Early Access Review’ were removed. Next, rows with reviews containing filtered content were removed, as Steam replaced sensitive words with the symbol ‘♥’, leading to incomplete comments. Figure 2 displays an example of Steam's automated filtering. Rows with comments containing merely whitespaces were moved also. Finally, rows with less than 20 characters were removed to remove comments that contained too short, difficult-to-interpret content. After performing the mentioned data cleaning procedure, the number of rows in the dataset was reduced to 3.95M, a 38.4% reduction in terms of size.

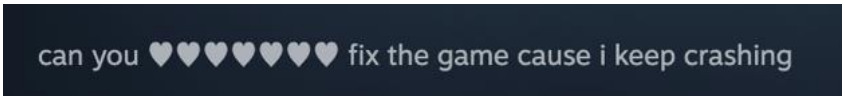



*Figure 2: Steam automated comment filtering. The sensitive word was replaced by consecutive heart symbols.*

#### 3.1.1.4 Exploratory Data Analysis

Exploratory data analysis (EDA) was performed on the preprocessed dataset before model training. EDA is concerned with finding information from raw data and generating informal conclusions about the data. It aims to think about the data from various points of view by utilizing an array of tools, such as analyzing statistical values and graph plotting Fields(Morgenthaler, 2009). First, the ratio between the number of positive reviews and negative reviews was calculated to be 5.14: 1 (See Figure 3), which was not surprising as similar ratios were recorded in a previous study across games in multiple genres (Guzsvinecz & Szűcs, 2023).

Next, the focus was shifted to analyzing the distribution of the number of words in reviews. After calculation with Python and Pandas, the medium number of words was 29, with a median number of 154 characters. It was suggested that our dataset contained comments with relatively shorter game reviews than a previous study, as the median number of characters was 25% smaller than a previous study of game reviews (Lin et al., 2019b). Also, 99% of reviews were 549 words or less, suggesting handling the majority of reviews will not be a computationally intensive task in terms of the length of a review. Further investigation regarding the distribution of the number of words in positive and negative sentiment reviews was conducted. It was discovered that negative reviews were longer than positive reviews, as the former had a median number of words equal to 40, while the latter was 27. The result also echoed the findings in the same previous study (Lin et al., 2019b).

Then, the focus was shifted to analyzing the common words of the reviews. Before analyzing, a further cleaning was performed, in which the flowchart of the process was shown below (See Figure 4). We first removed any hyperlinks and special markups (like ">", "&quot", "<p>"), in the comments. Then we removed any emojis in the sentences. Next, we convert all letters to lowercase and unify consecutive whitespaces to a single whitespace character. Then we remove any punctuation except ",", "." and "!". Finally, we performed stopword removal and stemming using NLTK for each review. Stopwords are a set of words that are ubiquitous yet carry little meaning to the text, such as 'a', 'I', 'do', 'be', 'then', 'that', and 'so'. Removing them can reduce the noise in the text. Stemming is a technique to reduce words in multiple-word forms to their base form. Applying the technique can reduce the feature space of words, avoid redundancy, and lead to a more consistent representation of the text. A. After that, the number of appearances of each word was calculated and the top 20 common words in the whole dataset were discovered. It was shown that the words were about the game, such as the word 'game', 'play', 'one', and 'story', and feeling towards the game, such as 'like', 'good', 'fun', 'love' (See Figure 5 (a)). If analyzing the sets with only positive or negative comments, 15 out of 20 frequent words in either positive or negative comments were in common (See Figure 5 (d)), showing huge overlapping in the set of most common words in the set of positive-only reviews and negative-only reviews.

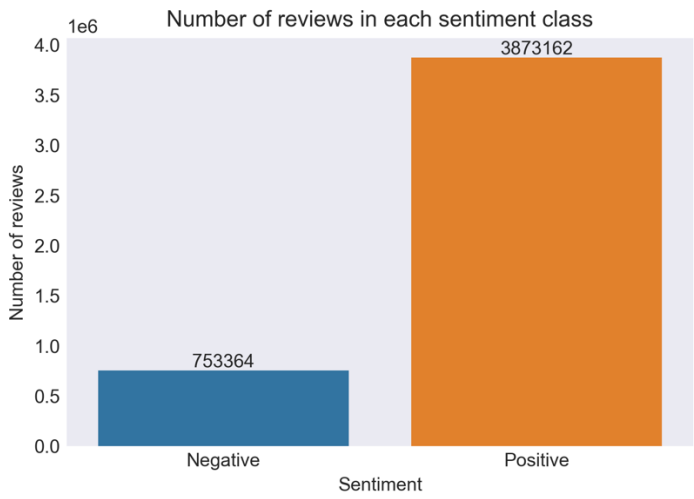



*Figure 3: Number of reviews in each sentiment class in the cleaned dataset with a 4.89: 1 positive to negative ratio.*

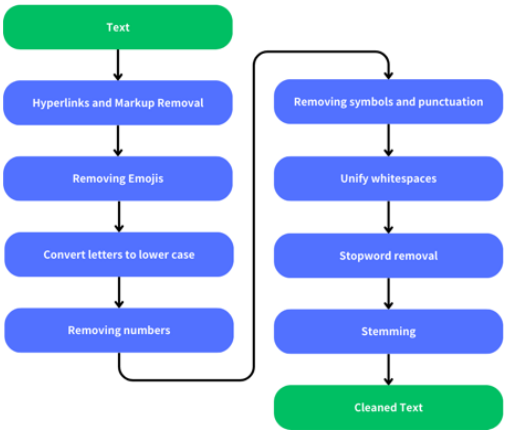



*Figure 4:Procedure of further data cleaning on the cleaned dataset.*

| | Common_words | count |
|---|---|---|
| 0 | game | 7200481 |
| 1 | play | 2106786 |
| 2 | like | 1474780 |
| 3 | get | 1224335 |
| 4 | one | 1034994 |
| 5 | good | 1000544 |
| 6 | time | 974589 |
| 7 | fun | 952969 |
| 8 | great | 849592 |
| 9 | realli | 845529 |
| 10 | stori | 737545 |
| 11 | make | 731250 |
| 12 | would | 662292 |
| 13 | even | 612833 |
| 14 | much | 558775 |
| 15 | well | 512212 |
| 16 | go | 511252 |
| 17 | love | 499568 |
| 18 | feel | 493996 |
| 19 | best | 490467 |

(a)

| | Common_words | count |
|---|---|---|
| 0 | game | 5759277 |
| 1 | play | 1718679 |
| 2 | like | 1168540 |
| 3 | get | 940747 |
| 4 | one | 852344 |
| 5 | good | 842329 |
| 6 | fun | 837099 |
| 7 | great | 775291 |
| 8 | time | 760950 |
| 9 | realli | 673990 |
| 10 | stori | 635701 |
| 11 | make | 573899 |
| 12 | would | 517651 |
| 13 | best | 459759 |
| 14 | love | 451082 |
| 15 | much | 440242 |
| 16 | well | 437956 |
| 17 | even | 426026 |
| 18 | go | 389224 |
| 19 | feel | 384278 |

(b)

| | Common_words | count |
|---|---|---|
| 0 | game | 1441204 |
| 1 | play | 388107 |
| 2 | like | 306240 |
| 3 | get | 283588 |
| 4 | time | 213639 |
| 5 | even | 186807 |
| 6 | one | 182650 |
| 7 | realli | 171539 |
| 8 | good | 158215 |
| 9 | make | 157351 |
| 10 | would | 144641 |
| 11 | go | 122028 |
| 12 | much | 118533 |
| 13 | buy | 117896 |
| 14 | want | 116730 |
| 15 | fun | 115870 |
| 16 | tri | 112440 |
| 17 | feel | 109718 |
| 18 | bad | 107320 |
| 19 | thing | 104981 |

(c)

| | Common_words | count_x | count_y |
|---|---|---|---|
| 0 | game | 5759277 | 1441204 |
| 1 | play | 1718679 | 388107 |
| 2 | like | 1168540 | 306240 |
| 3 | get | 940747 | 283588 |
| 4 | one | 852344 | 182650 |
| 5 | good | 842329 | 158215 |
| 6 | fun | 837099 | 115870 |
| 7 | time | 760950 | 213639 |
| 8 | realli | 673990 | 171539 |
| 9 | make | 573899 | 157351 |
| 10 | would | 517651 | 144641 |
| 11 | much | 440242 | 118533 |
| 12 | even | 426026 | 186807 |
| 13 | go | 389224 | 122028 |
| 14 | feel | 384278 | 109718 |

(d)

*Figure 5: Number of appearances of top 20 frequent words in the cleaned dataset after further data cleaning. (a): Words in the cleaned dataset with both positive and negative reviews. (b): Words in the cleaned dataset with only positive reviews. (c): Words in the cleaned dataset with only negative reviews. (d): Common words in (b) and (c).*

### 3.1.1.5 Datasets Preparation

Eight datasets were constructed from the cleaned dataset after Text Preprocessing. Two of them were for validation, and the remaining were for model training.

Regarding validation datasets, a balanced dataset and an imbalanced dataset were created to effectively perform cross-model performance comparisons and evaluate the performance of models in real-life situations. The balanced dataset contained 268588 reviews with the same number of positive and negative reviews, while the imbalanced dataset contained 790655 reviews with the ratio of the number of positive and negative reviews approximated to 5: 1, similar to the cleaned dataset. The creation process was as below. First, the imbalanced dataset was created by selecting 20% of the reviews randomly. These selected reviews were then dropped from the cleaned dataset. Next, the balanced dataset was created by first selecting 20 percent of the total number of negative reviews in the cleaned dataset, and then randomly selecting the same number of positive reviews from the cleaned dataset. Same as before, these selected reviews were dropped before the creation of training datasets.

Regarding training datasets, six training datasets were created to compare the performance of different models trained with different training dataset sizes and class distribution. Three different sizes were selected, which were 120K, 240K, and 480K. Then, for each size, a balanced and an imbalanced dataset was created. The ratio between the number of positive and negative was exactly 5.14: 1 in the imbalanced datasets. First, a balanced 120K dataset and an imbalanced 120K were created by random sampling without replacement. Then, each subsequent datasets in balanced and imbalanced categories were treated by doubling the number of training instances. The approach was analogous to the procedure in (Prusa et al., 2015). It was mentioned that with each smaller dataset being a subset of a larger dataset, a more meaningful comparison between performance and dataset size can be achieved, as any change in performance is the result of additional reviews instead of randomly selecting a completely new set of reviews (Prusa et al., 2015).

#### 3.1.1.6 Feature Extraction and Model Selection

Feature extraction is a fundamental and indispensable task in sentiment classification as it directly influences performance (Birjali et al., 2021). It extracts valuable information that describes the characteristics of the text from the words (Birjali et al., 2021). Birjali et al (2021) identified two major representations of features, which were Bag-of-Words (BoW), and Distributed Representation (also called Word Embedding).

BoW first creates a vocabulary of all unique words occurring in the document, then encodes a sentence as a vector with the length of the vocabulary of known words. The value of each position in the vector represents a count or frequency of the word in the vocabulary. However, BoW is incapable of representing the syntactic information of the text as it does not consider word order, sentence structure, or grammatical construction. (Birjali et al., 2021). Distributed Representation distributes the information of a word in a vector space with a fixed dimension where each word can be represented by a vector. Relation between the semantic meaning of words can be represented with vector operation. A famous example is that the result of *vector("King") – vector("Man") + vector("Woman")* is closest to the vector representation of the word "Queen" (Mikolov et al., 2013).

Considering the difference in feature extraction, three methods of feature extraction and the corresponding model were selected to compare the performance of different feature extraction and representation methods in sentiment classification on game reviews.

The first model, TFIDF-RF, applied Term Frequency-Inverse Document Frequency (TF-IDF) and Random Forest Classifier. TF-IDF is an example of BoW feature extraction. It measures the importance of a word to a corpus by considering its frequency in the document, which is a review, and its rarity in the whole corpus, which is all reviews in the training dataset. TF-IDF value can be calculated by first calculating the Term Frequency (TF) of term *t* within a document *d*, where $f_{t,d}$ represents the frequency of the term *t* in document *d* (Eqt. 1). Then the Inverse Document Frequency (IDF) of term *t* within the whole corpus *D* was calculated, in which the numerator represents the total number of documents, and the denominator represents the document frequency of term *t* (Eqt. 2). An extra '1' in both numerator and denominator equation (2) was to prevent zero divisions. Lastly, multiplying TF and IDF results in TF-IDF (Eqt. 3). TF-IDF assigns higher weights to words that occur frequently in a document but are rare in the corpus, allowing models to prioritize terms that are more indicative of sentiment in each review. Random forest was selected because of its ensemble learning approach, as multiple decision trees are combined to make predictions.

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{1}$$

$$idf(t,D) = \log \frac{1 + |D|}{1 + |\{d \in D : t \in d\}|} \tag{2}$$

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D) \quad (3)$$

The second model, GloVe-CNN, applied Global Vectors for Word Representation (GloVe) and Convolutional Neural Network (CNN). GloVe is an example of distributed representation. It captured the global corpus statistics directly by computing the word vectors based on the probability of the appearance of a target word given a context word, which is the co-occurrence probabilities of words. It first computed a large co-occurrence matrix based on the number of vocabulary items in the corpus, then optimized the word vector representation with least square regression with a custom loss function. Unlike TF-IDF, it encodes semantic relationships between words, which allows models to understand contextual information and capture sentiment nuances. CNN was selected because of its frequent use in sentence classification with ongoing advancements (Fachrul et al., 2022; Li et al., 2022; Maisa et al., 2023; Wenxuan & Yuxuan, 2022). It treats a list of word vectors from the sentence like an image with size *(d, w)*, where *d* = dimension of each word vector, *w* = length of a sentence with padding, if necessary. Then the filters learn the word features by adjusting the weights of each element in the kernel matrix.

The third model, BERT, was a fine-tuned model from pre-trained Bi-directional Encoder Representation from Transformer (BERT). Although embedding each word in the form of distributed representation, the model has some fundamental differences compared to the first two models. First, regarding the nature of the embedding vector, the embedding from BERT contains more information than previously mentioned distributed representations, such as word2vec and GloVe. In the second model where GloVe embedding was applied, a static vector was used to represent the target word regardless of the context words around itself. However, large language models, such as Generative Pre-trained Transformer (GPT), BERT, and ELMo, used contextualized embeddings, in which different embedding vectors were used for the same word in different contexts. Indeed, a recent review confirmed that BERT token representations contain both syntactic and semantic information fields (Rogers et al., 2021). Second, regarding the training method, this model trained with a larger corpus with a larger number of words. Unlike the first two models which a model is trained directly and only from the corpus of the dataset of the task, this model adopted the approach of fine-tuning for a downstream task, such as text classification, translation, and question-

answering, from a pre-trained language representation in a language model, such as GPT, and BERT. It allowed a larger model to learn a universal representation of words that can transfer to a wide range of tasks with little adaptation (Radford et al., 2018). Third, in the second model, given a target word represented by its word vector, it only considered a fixed window size of the context word due to the fixed kernel size in convolution operation in CNN. However, thanks to the self-attention mechanism in the Transformer, a target word can consider any word within the document and assign a different value. Last, BERT introduced bi-directional pretraining instead of using uni-directional pretraining in GPT. It enabled the ability of a given word to consider context words in both directions, which was believed to be more powerful than a uni-directional model or a shallow concatenation of left-to-right and right-to-left models (Devlin et al., 2019). For the conciseness of the report, the technical structure and computational details of BERT were omitted, to which the corresponding details can be referred (Devlin et al., 2019).

#### 3.1.1.7 Model Implementation

Regarding the first model, a training pipeline was implemented containing three major components. First, the top *N* words with the highest TF score were selected to form the vocabulary. Next, the TF-IDF representation of these selected words was calculated. Each sentence was then transformed into a vector with each element representing the TF-IDF value of each word. Finally, the vectors were fed to train a Random Forest classifier a group of bootstrapped classification trees. The first model was implemented using scikit-learn.

Regarding the second model, a training pipeline consisting of an embedding matrix and a CNN model similar to the structure in (Kim, 2014) was implemented. Before entering the pipeline, the top N words with the highest frequency across the whole training dataset were selected. Also, the length of a sentence was limited to a fixed length *L* to prevent extended calculation time, and padding would be applied if necessary. In the pipeline, the embedding matrix was used to construct vector embedding of words in the sentence. The embedding matrix was set to be non-static, and the embedding vectors were updated during training to grasp the semantic meaning of words in the context of game reviews. It was reported that CNN models trained with non-static word vectors uniformly outperformed those trained with static word vectors (Zhang & Wallace,

2016). The CNN model consisted of three convolutional layers, placed in a flat structure. However, unlike convolutional layers in image-related tasks where a square kernel matrix will go through the whole image in both directions, the kernel will look into a window of a fixed number of words to produce a new feature. Then, each layer was followed by a 1D max pooling layer to select the most important feature in each channel of each convolutional layer. The features were then concatenated to a single-dimension array, and dropout was applied to the array as regularization. Finally, a fully connected layer is applied after the dropout layer to produce classification results. In implementation, instead of assigning a different filter size to each convolutional layer, we followed the recommendation of assigning all three convolutional layers with the same filter size (Zhang & Wallace, 2016), as it resulted in better performance than combining different sizes with the default setting. Original L2 weight regularization was omitted for simplified implementation. The resulting model is shown in the figure below (See Figure 6). The model was implemented using Keras, a high-level abstraction library for building deep learning models created by Google.

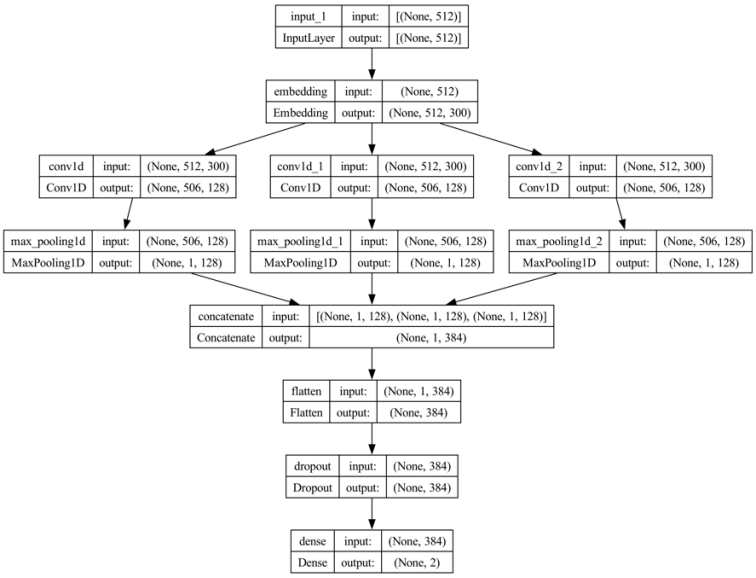



*Figure 6: Model structure of CNN in model GloVe-CNN.*

Regarding the third model, a training pipeline consisting of a tokenizer and a pre-trained BERT model was implemented. $BERT_{BASE}$ was selected instead of $BERT_{LARGE}$ as the former contained only about 1/3 of the parameters while achieving state-of-the-art results compared to other models (Devlin et al., 2019). The reduced parameters also significantly reduced the fine-tuning time required with a large training dataset on personal-scale hardware. Regarding the tokenizer, a pre-trained case-sensitive tokenizer was selected, as capitalized words often contain more extreme emotions. Truncating and padding were applied to each review, as there is a maximum length of

tokens the model can receive as input, which is 512. Regarding the model, a pre-trained $BERT_{BASE}$ paired with the case-sensitive tokenizer was initialized. The training pipeline was implemented with HuggingFace, a high-level abstraction library for building transformer models.

#### 3.1.1.8 Model Training

Before training, further data cleaning customized with each model was performed on each training dataset. Applying customized further data cleaning for each model is necessary as it allows features to be consistent with the different feature extraction methods of each model, enhancing their performance. Flowcharts of three further data-cleaning processes are presented below (See Figure 7). Regarding the first model, the further data cleaning process following the same as that before analyzing the common words of the reviews was applied. The reasons behind applying the further data cleaning process were to reduce the feature space of words, avoid redundancy due to various verb forms, and lead to a more consistent representation of words in the corpus. Regarding the second model, the same further data cleaning process was applied to the training dataset except for stopword removal and stemming, as no stopword removal and stemming were performed during the pretraining GloVe. Regarding the third model, only the removal of emojis, hyperlinks, and markups was performed. Punctuations and numbers were retained as pre-defined tokens were available in the tokenizer of BERT. Also, punctuations were necessary for BERT to separate different sentences using special preserved tokens defined in the tokenizer. Both stemming and stopword removal were not applied as the tokenizer can break down words in various forms to one or more than one token which represented the stem and the verb form.

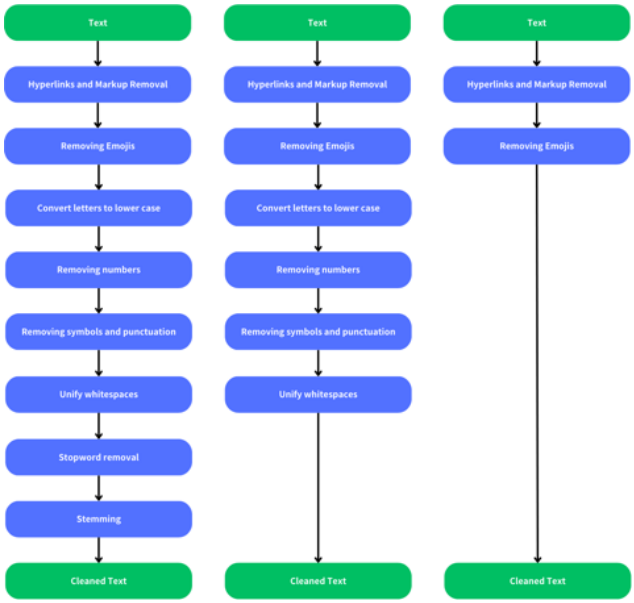



*Figure 7: Customized further data cleaning to each model. Left: TFIDF-RF. Middle: GloVe-CNN. Right: BERT*

For model training with all different training datasets, 10% of the training dataset will be used as a testing dataset during training to monitor the training process.

Regarding model parameters in the first model, $N$ was set to 20K, which was chosen to represent non-arcane vocabs without posing significant computational difficulty in the reviews. The number of classification trees was 100. Other parameters of components used in the pipeline followed the default parameters in scikit-learn.

Regarding the model parameters in the second model, $N$ was set to 20K, and $L$ was set to 512, which was identical to the maximum length of tokens in BERT. A GloVe representation pre-trained on 6 billion tokens, with 300-dimensional word vectors, was used to initialize the word embedding layer. Each convolutional layer contains 128 filters, with filter size = 7. Dropout probability was set to 0.3. The batch size was set to 128 to fully utilize the memory of the GPU while presenting a more stable gradient. The loss was computed using sparse categorical cross-entropy. Adam optimizer was applied with learning rate = 1e-3, $\beta_1 = 0.9$, $\beta_2 = 0.999$. Other parameters of components used followed the default parameters in Keras. To prevent overfitting, the learning rate was reduced by 0.8 when there was no reduction in testing loss during training. Early stopping was performed when there was no reduction in testing loss for 5 consecutive epochs. After training, the best model with the lowest testing loss will be stored and used for evaluation.

Regarding the model parameters of the third model, a pre-trained $\text{BERT}_{\text{BASE}}$ (Devlin et al., 2019) model was used for further fine-tuning. Following the recommendations of (Devlin et al., 2019), the batch size was set to 32. Adam optimizer was used with

learning rate = 2e-5, as a higher learning rate will result in catastrophic forgetting shown by (Sun et al., 2019), $\beta_1$ = 0.9, $\beta_2$ = 0.999. L2 weight decay was set to 0.001. No learning rate warmup was applied as the number of steps in training with 120K datasets was fewer than 10000. The number of epochs was set to 3. Other parameters of components used followed the default parameters in HuggingFace. The best model with the lowest testing loss will be restored and saved for evaluation.

#### 3.1.1.9 Model Evaluation

The evaluation was performed with the view to answering the research questions, and therefore, selecting the best performance model for deployment. Weighted average F1-score was chosen for the chief performance metric as the equation considered both recall and precision by calculating a harmonic mean of them. To define a weighted average F1-score from F1-score, we first let there be $n$ sentiment classes and define $y_i$ to be the number of samples within the sentiment class $i$. Then we calculate the precision and recall for each sentiment class $i$. Next, we calculate a weighted average of the precision (See Eqt. 4) and recall (See Eqt. 5) of all sentiment classes $n$. Finally, we calculate the weighted average F1-score by considering the weighted average of precision and recall (See Eqt. 6).

$$WeightedAveragePrecision = \frac{\sum_{y=1}^{n} y_i \frac{TP_i}{TP_i + FP_i}}{\sum_{y=1}^{n} y_i} \tag{4}$$

$$WeightedAverageRecall = \frac{\sum_{y=1}^{n} y_i \frac{TP_i}{TP_i + FN_i}}{\sum_{y=1}^{n} y_i} \tag{5}$$

$$WeightedAverageF1 = \frac{2}{\frac{1}{WeightedAveragePrecision} + \frac{1}{WeightedAverageRecall}} \tag{6}$$

#### 3.1.1.10 Model Deployment

Before deploying on the virtual machine, the trained model was converted to ONNX format for fast CPU inference with ONNXRuntime, as a previous study showed that a 36.5% reduction in inference execution time was recorded on ONNX + ONNXRuntime than Torch Script + Libtorch, a library for running PyTorch models on C++ (Öğüt, 2021). Comparisons of end-to-end inference times of a prediction on CPU between the

original and converted ONNX model were drawn on two different machines, one on a moderate window machine running WSL 2 and another on an Apple laptop. An end-to-end inference refers to the inference from a pre-processed review, then producing possibilities on both sentiment classes. WSL2 was used instead of running natively on Windows as TensorFlow, the backend of Keras, stopped complete support on native Windows. The specifications of the two machines are listed below.

| | Machine 1 | Machine 2 |
|---|---|---|
| CPU | Intel Core i5-8250U,<br>4 Cores, 8 Threads | Apple M1 Max, 8 Performance Cores,<br>2 Efficiency Cores |
| RAM | 8GB (in WSL 2) | 32GB |
| OS | Ubuntu 22.04 (in WSL 2)<br>Windows 10 22H2 | macOS Monterey 12.6.2 |

Python was used to run the measurement program as it was the programming language for both ML development and deployment. For the sake of completeness, the version information of used software is provided below.

- Python 3.9.18
- scikit-learn 1.3.0
- TensorFlow 2.15.0
- Keras 2.15.0
- torch (PyTorch) 2.1.0
- transformers (HuggingFace) 4.35.0
- accelerate (HuggingFace) 0.24.1
- onnx (ONNX) 1.14.1
- onnxruntime (ONNX Runtime) 1.16.3
- skl2onnx 1.16.0
- tf2onnx 1.15.1
- optimum (HuggingFace) 1.16.1

Regarding the conversion progress, an end-to-end ONNX model was generated from scikit-learn. However, the text vectorizer component in Keras and the tokenizer component in HuggingFace were unable to convert to ONNX as ONNX does not support string manipulation. Therefore, original components from Keras and HuggingFace were used in the end-to-end ONNX inferencing.

For the measurement program, after loading the models and reviews, a warmup of inferencing 1000 reviews with batch size = 1 was performed. It ensures the model is fully loaded to the memory, reducing variance in inference time. Then, the time of inferencing 2000 reviews with batch size = 1 was measured. Batch size = 1 was selected as the sentiment classification result should be provided as soon as the review was posted to the platform, in which the message was consumed immediately by the Python NLP backend. The time required for performing data cleaning to the reviews was unrecorded as it is insignificant to the overall inference time. The times of inferencing each review were then stored for further analysis.

### 3.1.2 Topic Modeling

#### 3.1.2.1 Topic Modeling Application

Topic Modeling is an unsupervised machine learning technique that aims to discover hidden themes in textual data and perform categorization (Churchill & Singh, 2022). Applying topic modeling for game reviews benefits both potential players and developers. To potential players, grouping reviews allows them to quickly glance at a specific aspect of a game, such as graphics, compatibility, and gameplay, without the need to read hundreds of fewer related comments. This shortens the purchasing decision-making progress. To developers, topic modeling helps them to better prioritize tasks to be done to cater to their players' needs. It distinguishes reviews based on various aspects of the game, such as graphics, gameplay, and bug reports, selecting contributing comments from unhelpful, emotional comments. This empowers developers to quickly locate issues that players complain about the most, for instance, game-blocking bugs and crashes (Lin et al., 2019b), and reallocate manpower and time to provide a more satisfactory gaming experience to both current and future players.

#### 3.1.2.2 Dataset Creation

The data used for topic modeling will be current games listed on the Steam Platform and a list of genres selected from the available genres list on Steam the reviews scraped are then linked back to the genre associated with the game for EDA and to the game based on the gameId stored in our database. For games that do not exist in our database, the reviews are not considered and ignored.

In total, 3 different datasets will be created, one containing all reviews, one for the action genre, and one for the indie genre which accounts for more than half of all total reviews listed on the Steam platform.

#### 3.1.2.3 EDA

Exploratory Data Analysis revealed that the Action and Indie genres accounted for more than half of the total reviews in the filtered dataset, at 2.05 million out of the 4.05 million (See Table 1). Given a game can have more than one genre or description tag associated, the total size of this dataset is significantly larger than the actual size of review dataset. With the majority of the reviews being for games with Action or Indie tags, these two genres will be the main focus of the topic modeling effort.

| Genre/Description | Number of Reviews |
|---|---|
| Action | 1314407 |
| Indie | 741913 |
| Adventure | 636492 |
| RPG | 545025 |
| Strategy | 409708 |
| Simulation | 260079 |
| Free to Play | 246372 |
| Casual | 209223 |
| Massively Multiplayer | 94777 |
| Racing | 25863 |
| Spirts | 24616 |

*Table 1 Distribution of reviews based on genres/description out of the top 10 genre/description.*

#### 3.1.2.4 Model Selection

Three topic Modeling techniques were selected to apply to the task of topic modeling on game reviews. In particular, LDA, Contextualized Topic Model (CTM) (Bianchi et al., 2021), and BERTopic (Grootendorst, 2022) were selected. LDA was selected because of its generality and frequent usage in various topic modeling problems (Churchill & Singh, 2022). It serves as a baseline to compare existing results in examined previous studies. BERTopic and CTM were selected as

contextualized embeddings were used for building topic models in both techniques, in which contextualized embeddings produced the best performance among the three examined feature extraction methods in the task sentiment analysis.

LDA is a probabilistic, bag of words model that represents a topic based on the probability of appearance of words in that topic. It assumed each word in the document is created by sampling a topic from the distribution of topics for the document, and then sampling a word from the topic (Abdelrazek et al., 2023). The algorithm aims to find the topic-word distribution that maximizes the likelihood of documents in the dataset over *K* number of topics (Churchill & Singh, 2022). Two more parameters, alpha, and beta, were used to define Dirichlet priors for drawing topic distribution and word distribution within a topic.

CTM is a bag of words model extended from a neural topic model. A neural topic model is an encoder-decoder that first maps the bags-of-word (BoW) document representation to a continuous latent representation through the encoder, then reconstructs the BoW by generating the words from the latent representation through the decoder (Bianchi et al., 2021). An example is ProdLDA (Srivastava & Sutton, 2017)., which improves from LDA by approximating the Dirichlet prior in LDA using Gaussian prior and replacing the distribution of words with the product of experts (Srivastava & Sutton, 2017). CTM extends ProdLDA by adding contextualized embedding generated from SBERT (Reimers & Gurevych, 2019).

Unlike the first two models which represent a topic by word distribution, BERTopic adopted a clustering embedding approach with four key steps. First, similar to the second model, contextual document embeddings are generated from SBERT. Then, the dimensionality of embeddings is reduced using UMAP and clustering is performed using HDBSCAN. Next, a class-based TF-IDF (c-TFIDF) was used to model the importance of words in a cluster to generate topic-word distribution. Finally, topic merging was performed based on the c-TFIDF representation to reduce the number of topics to a specific value.

#### 3.1.2.5 Model Evaluation

To evaluate the performance of different topic models, both quantitative and qualitative approaches were planned to be applied. Regarding qualitative analysis, visualization tools for all three topic modeling techniques were identified. Specifically, pyLDAvis will be used to analyze the result of LDA qualitatively, while visualization tools, including Distance Map, were identified for the remaining two models. Regarding quantitative analysis, measurement in terms of topic coherence and topic diversity will be taken using pre-defined metrics.

### 3.1.3 Keyword Extraction

Keyword extraction is a technique to extract a set of keywords from a document without manual work (Khan et al., 2022). It can be applied to game reviews to extract critical information, especially for longer reviews, and to provide a high-level overview of the review content and sentiment. It can also act as a topic modeler to support the topic modeling tools by assigning interpretations to the topics categorized by the topic modeling models.

Regarding Keyword Extraction, the

Despite the existence of various developed keyword extraction models, such as KeyBERT, YAKE, Text Rank, and Page Rank, none of them produce an interpretable and easy-to-read description of a topic after topic modeling is applied. For instance, in Figure 8, KeyBERT generates short n-gram keywords while Llama2 can summarize the topic keywords into a single term. Therefore, our focus was shifted to using pre-trained Large Language Models (LLMs) to produce comprehensible descriptions of identified topics in the section Topic Modeling.

| | Topic | Count | Name | CustomName | Representation | KeyBERT | Llama2 | MMR | Representative_Docs |
|---|---|---|---|---|---|---|---|---|---|
| 0 | -1 | 34138 | -1_the_of_and_to | Computer Vision and Machine Learning | [the, of, and, to, in, we, for, is, that, on] | [models, model, accuracy, datasets, networks, ... | [Computer Vision and Machine Learning, , , , ,... | [the, of, and, to, in, we, for, is, that, on] | [ We propose a random convolutional neural ne... |
| 1 | 0 | 10326 | 0_policy_reinforcement_rl_agent | Deep Reinforcement Learning Challenges | [policy, reinforcement, rl, agent, learning, c... | [reinforcement, learning, robots, dynamics, ro... | [Deep Reinforcement Learning Challenges, , , , ,... | [policy, reinforcement, rl, agent, learning, c... | [Efficient exploration is a crucial challenge ... |
| 2 | 1 | 3574 | 1_privacy_federated_fl_private | Privacy-preserving federated learning | [privacy, federated, fl, private, clients, dat... | [federated, heterogeneity, decentralized, dist... | [Privacy-preserving federated learning, , , , ... | [privacy, federated, fl, private, clients, dat... | [ Providing privacy protection has been one o... |
| 3 | 2 | 3523 | 2_speech_audio_speaker_music | Audio-Visual Speech Separation and Recognition | [speech, audio, speaker, music, asr, acoustic,... | [convolutional, encoder, voice, speech, traine... | [Audio-Visual Speech Separation and Recognitio... | [speech, audio, speaker, music, asr, acoustic,... | [ In recent years, neural network based metho... |
| 4 | 3 | 2275 | 3_equations_differential_physics_neural | Solving Partial Differential Equations using N... | [equations, differential, physics, neural, pde... | [pdes, pde, modeling, dynamics, computational,... | [Solving Partial Differential Equations using ... | [equations, differential, physics, neural, pde... | [Physics-informed neural networks (PINNs) have... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 117 | 116 | 160 | 116_augmentation_mixup_data_training | Data Augmentation Techniques in Deep Learning | [augmentation, mixup, data, training, augmenta... | [adversarial, augmentation, imagenet, regulari... | [Data Augmentation Techniques in Deep Learning:... | [augmentation, mixup, data, training, augmenta... | [ Data augmentation techniques have become st... |
| 118 | 117 | 160 | 117_crowdsourcing_workers_crowd_worker | Crowdsourced Data Labeling | [crowdsourcing, workers, crowd, worker, crowds... | [crowdsourcing, crowdsourced, annotators, crow... | [Crowdsourced Data Labeling, , , , , , , , , ] | [crowdsourcing, workers, crowd, worker, crowds... | [ There is a rapidly increasing interest in c... |
| 119 | 118 | 155 | 118_summarization_summaries_summary_abstractive | Automated Document Summarization | [summarization, summaries, summary, abstractiv... | [summarization, summarisation, summarizing, su... | [Automated Document Summarization, , , , , , ,... | [summarization, summaries, summary, abstractiv... | [ Abstractive Text Summarization is the proce... |
| 120 | 119 | 152 | 119_design_circuit_circuits_chip | Design Automation for Analog Circuits using Re... | [design, circuit, circuits, chip, synthesis, d... | [circuits, circuit, analog, vlsi, optimization... | [Design Automation for Analog Circuits using R... | [design, circuit, circuits, chip, synthesis, d... | [ The design automation of analog circuits is... |
| 121 | 120 | 150 | 120_sentiment_aspect_analysis_polarity | Sentiment Analysis of Online User-Generated Co... | [sentiment, aspect, analysis, polarity, review... | [embeddings, sentiment, sentiments, annotated,... | [Sentiment Analysis of Online User-Generated C... | [sentiment, aspect, analysis, polarity, review... | [Targeted Sentiment Analysis aims to extract s... |

*Figure 8: Example output of BERTopic using KeyBERT and Llama2 to name the topics.*

Instead of using existing online services like OpenAI, or Azure ChatGPT, the utilization of locally deployed LLMs offers two main advantages. Firstly, by opting for local deployment, the privacy of user data is ensured, mitigating the risk of any confidential or sensitive information being leaked. Secondly, local deployment allows for the utilization of a wide range of models, including Llama2 from Meta, phi-2 from Microsoft, Mistral developed by Mistral AI, or even custom fine-trained models. This not only grants developers fine-grained control but also enables the use of "uncensored models" that are fine-tuned on datasets without filtered responses. This is particularly crucial in analyzing game reviews where the content may contain sexual, aggressive language that can trigger the model content filtering mechanism. Examples of game reviews that triggered the Azure ChatGPT content filtering mechanism are shown in Figure 9.


(a) (b)

*Figure 9: Two game reviews and the response from ChatGPT hosted by Azure when prompting to classify their sentiment.*

The results of the keyword extraction will be evaluated quantitively by manual inspection. In particular, the coherence between the generated name for topic c and some most representative documents of the topic will be examined.

## 3.2 Frontend Web Application

This section discusses the technologies that will be involved in developing the frontend system (Section 3.2.1), the user interface design approach and method for the web application (Section 3.2.2), and the proposed implementation of authentication process and user information access and management for the web application (Section 3.2.3).

### 3.2.1 Technologies Involved

First, the selected framework for developing the application is React, which is a widely adopted and popular web framework known for its extensive range of community-made packages, which adopt a declarative and component-based approach to building user interfaces. To enhance the appearance and functionality of the user interface.

Second, Material UI (MUI), a component library of React that provides a set of prebuilt and customizable UI components, will be used because it provides an efficient, production-ready, and complete set of components that can facilitate the website development tremendously. For instance, it provides input components such as the text field, select, and button components, which can be used to create different forms, such as the login form and the add review form.

Third, Next.js, a meta-framework built on top of React, has been chosen to provide support for modern features like Server-Side Rendering, File-based Routing, Secure Fetching, and Performance Optimization. These features enable the application to be maintainable, responsive, performant, and scalable.

Finally, TypeScript, a syntactic superset of JavaScript that offers high-level type safety, has been chosen as the programming language to enhance development efficiency and minimize unintended bugs caused by typing mismatch, as TypeScript ensures all variables only access authorized memory locations that are well-defined and permissible.

### 3.2.2 Design Approach

To cater to a broader audience and enhance accessibility for users across different devices and platforms, such as mobile, tablet, and desktop, the web application must be responsive and performant. Therefore, the web application will be designed under the Responsive Web Design (RWD) approach, which adjusts the size, position, and visibility of webpage elements based on the device viewport to ensure that the website will have a natural and intuitive appearance on different screen sizes, resolutions, and orientations. This design approach ensures that the web application is compatible with

various devices with different screen sizes. This will enable users to seamlessly interact with the platform regardless of their preferred device and platform, thereby expanding our reach and maximizing user engagement.

In addition, the design process of the web application is facilitated by Figma, a popular and collaborative design tool for application development. Figma offers valuable community assets for creating the web application prototype, such as the Material UI asset that contains all the prototype assets of the pre-built UI components provided by the Material UI library. In addition, Figma enables collaborative design features, which allow multiple people to join and participate in the design process as a team, making it appropriate for a group project.

### 3.2.3 Frontend Authentication

The frontend application's user authentication will be implemented using JSON Web Token (JWT) (See section 3.3.2), HTTP cookie (browser cookies), and React's *useContext* hook. The user login authentication will invoke one of the two backend APIs, depending on the presence of the refresh token in the HTTP cookies.

The first API, login, takes the user credentials as the request body and returns the access token and refresh token in the response body. This API is invoked only when the HTTP cookie does not contain a valid refresh token. The users can access this API through the login form in the web application, where they must enter their username or email and password. The login form has a "Remember Me" checkbox. If the user selects this option, the refresh token is stored as a persistent cookie with a 7-day expiration time. Otherwise, the refresh token is stored as a session cookie without an expiration time, and the cookie will expire if the user closes the browser session.

The second API, refreshToken, refreshes the session by generating a new access token. It takes the refresh token as the request body and returns a new valid access token in the response body. When the user accesses the web application, the frontend application will verify the existence of the refresh token in the HTTP cookies. If it exists, the application will invoke the API to refresh the session and obtain a new access token. The backend system will handle the invalid or expired refresh tokens by sending the appropriate error message in the response body.

Once the access token is obtained successfully by either method, it will invoke the *userAuth* API to fetch the updated user information. This API takes the access token as the request body and returns the user information in the response body if the access token is valid.

To manage and access the user information globally for the react application, the *useContext* hook will be employed. The user information will be stored in a mutable state using the *useState* hook. A context provider will be created with the user information state as a value. The page components will be wrapped within the provider to allow the provider to pass the user information state value to the pages. The page components can access the user information through the *useContext* hook via the context provider.

## 3.3 Backend Technologies

This section discusses the implementation of 9 services supporting the Backend solutions, including Spring Boot Server Application (Section 3.3.1), Authentication with JWT (Section 3.3.2), Email Service (Section 3.3.3), Database (Section 3.3.4), Object Storage (Section 3.3.5), Continuous Integration/Continuous Delivery (Section 3.3.6), Message Queue (Section 3.3.7), Hosting (Section 3.3.8) and Monitoring (Section 3.3.9).

### 3.3.1 Spring Boot Server Application

The backend system will be built using the Java Spring Framework, a common framework used to build high-performance and scalable Enterprise Application Programming Interface (API) solutions.

Spring Boot provides most of the functionality needed for a scalable backend API system, including security, MVC (Model View Controller), Batch Processing, High Performance, and non-blocking event loops. All data access requests will be made to the backend system through RESTful Hypertext Transfer Protocol (HTTP) requests. Most business logic and validation will only be performed in the backend system to provide security and high performance through parallelization on clusters as it has linear scalability.

By centralizing these processes in the backend system, parallelization can be leveraged on clusters for improved performance. Furthermore, this approach allows for linear scalability, ensuring that the system can efficiently handle an increase in demand.

External libraries that are used in the application will be installed with Apache Maven, an open-source tool for building and managing any Java-based project. The deployed server will use Maven to generate an executable JAR file using the production environment.

### 3.3.2 Authentication with JWT

The backend system implements the authentication mechanism based on the JSON Web Token (JWT) RFC standard, using Spring Security as the framework for authentication and access control. The system stores the tokens in both the client-side browser and the server-side database for security and convenience. When a user authenticates successfully, the system returns a JWT response to the client, which contains the user information and two tokens: the Access Token and the Refresh Token. The client utilizes these tokens to obtain authorization for API calls and to refresh the session when needed. The system ensures the security and validity of the authentication by applying a digital signature to the token data.

The system adopts HS256 as the encryption algorithm for the JWT signature, which is a symmetrical algorithm that relies on a shared secret key between the identity provider (Spring Security) and the application user. HS256 offers an adequate level of security and high performance for the system, as the system is the sole consumer of the JWT. An alternative algorithm, RS256, is an asymmetrical algorithm that uses a public and private key pair to generate and verify the JWT signature. RS256 is more suitable for scenarios where the client is not controlled by a single platform or application, as the client only needs to know the public key.

The system only includes non-sensitive user information in the Access Token, which can be decoded and viewed by using a public JWT reader (See Figure 10). The system sends the user's ID, email, and name as the user claims, along with the "exp" (Expired At) and "iat" (Issued At) claims, which indicate the expiration and issuance time of the token. The client can use these claims to determine when to renew the token or to prompt the user to log in again. Information can be added to the claims easily to support future

development by modifying the HashMap used to generate the Token without affecting Authentication.

```
{
  "email" : "u3578552@connect.hku.hk",
  "id" : 39,
  "exp" : 1703480516,
  "name" : "JackyLee997",
  "sub" : "u3578552@connect.hku.hk",
  "iat" : 1703394116
}
```

*Figure 10 JWT Claims extracted from the Access Token user received on login*

To reduce the storage cost of authentication and refresh tokens, whenever a user logs in, stale tokens older than 2 days will be deleted from the database (See Figure 11).

```
@Modifying
@Query(value = """
    delete from Token t where t.createdAt < :date
  """)
void deleteAllByCreatedAtBefore(Date date);
```

*Figure 11 Function and SQL that remove stale and outdated authentication and refresh token*

### 3.3.3 Email Service

Email service will be set up to support User authentication services, including email verification and forgotten passwords. We will utilize the Gmail SMTP Server to send the email from the Google account created. Utilizing the Google SMTP Server instead of a separate Mail server reduces the workload of our virtual machine and improves the efficiency of the User registration and authentication workflow.

All emails will be sent from Spring Boot using the Spring Email library to help establish the connection to the SMTP server and provide detailed results and tracking of the email sent without additional setup and configuration.

Gmail offers free email service with the additional advantage of reducing the email being flagged as Spam email or filtered compared to using an email attached to a custom domain.

### 3.3.4 Database

MySQL will be adopted as the preferred database due to its ability to handle relational data effectively and deliver high performance. Digital Ocean, a reputable cloud service provider, has been chosen to host the database.

To streamline database access and optimize Create, Read, Update, and Delete (CRUD) operations, Java offers the Java Persistence API (JPA). By leveraging the Spring JPA library, the application can benefit from improved performance and reduced boilerplate code for database interactions. JPA also encompasses features that support the Atomicity, Consistency, Isolation, and Durability (ACID) model, thereby ensuring data integrity and consistency when deployed as a distributed system. To ensure a structured and organized approach to database management, the creation and management of database tables and entities will be handled exclusively by Spring JPA. This approach guarantees the maintenance of data integrity and consistency throughout the lifecycle of the application. The entity relation diagram encapsulates the data necessary for the platform (See Figure 12), such as game, reviews, and user data and the attributes present in each entity. It also shows the relation and cardinality between entities and their representation in a database schema.

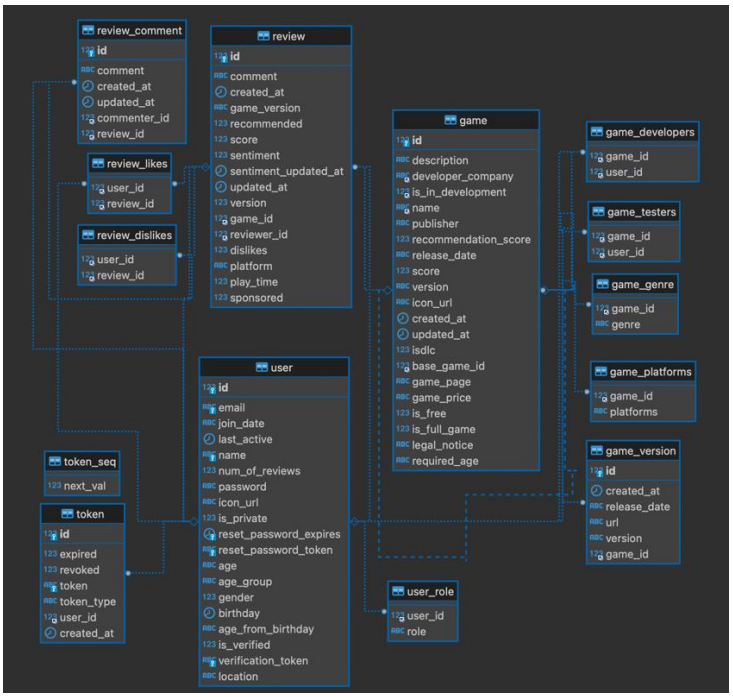



*Figure 12 Database Entity Relations Diagram*

Sensitive User data including Passwords and tokens will be encrypted before being stored in the database to ensure data security and will not be serialized and sent to users.

### 3.3.5 Object Storage

The project will incorporate a Simple Storage Service (S3) compatible storage solution to house all files provided by Digital Ocean. This includes but is not limited to, text documents, images, videos, and audio files. By opting for an S3-compatible storage bucket, the application can leverage the capabilities of the Amazon Web Service (AWS) Software Development Kit (SDK) to securely access, upload, and remove files from the bucket (See Figure 13).

```
public String uploadFile(final String fileName, final MultipartFile file) {
    try {
        ObjectMetadata metadata = new ObjectMetadata();
        metadata.setContentLength(file.getSize());
        metadata.setContentType(contentType(file));
        metadata.setHeader("x-amz-acl", "public-read"); // publicly accessible, comment this to not publicly accessible
        PutObjectResult result = amazonS3Client.putObject(bucketName, fileName, file.getInputStream(), metadata);

        System.out.println("Content - Length in KB : " + result.getMetadata().getContentLength());

        return result.getETag();
    } catch (IOException ioe) {
        logger.error("IOException: " + ioe.getMessage());
    } catch (AmazonServiceException serviceException) {
        logger.info("AmazonServiceException: " + serviceException.getMessage());
        throw serviceException;
    } catch (AmazonClientException clientException) {
        logger.info("AmazonClientException Message: " + clientException.getMessage());
        throw clientException;
    }
    return null;
}
```

*Figure 13 Sample Code to upload a file to the S3 Bucket*

The integration of an S3-compatible storage service offers a multitude of advantages. Primarily, it ensures high availability, thereby facilitating continuous access to the stored files. This is critical to guarantee uninterrupted service to the users. Furthermore, it provides high scalability, effectively accommodating the potential growth of the application and its associated file storage needs. This scalability ensures that the application remains future-proof and can handle increased demand efficiently. Lastly, the use of such a service guarantees high-performance file access and upload capabilities. This enhances the efficiency of file-related operations, thereby improving the overall user experience. **The dashboard** provided by Digital Ocean allows for clear analysis, modifications, and an overview of the files stored in the bucket (See Figure 14).


*Figure 14 Digital Ocean Spaces Dashboard*

### 3.3.6 Continuous Integration/Continuous Delivery (CI/CD)

Continuous Integration/Continuous Deployment (CI/CD) is a transformative approach to software development that emphasizes automated testing and deployment, facilitating rapid iteration and robustness of applications and services. The CI component ensures that as developers integrate code changes, automated tests are run, catching issues early and often, which is far more efficient than the traditional method of late-stage testing. This frequent integration helps maintain code quality and accelerates the development process.

Continuous Deployment builds on this by automatically deploying the code changes to production after successful testing, allowing for immediate use and feedback. This eliminates the need for manual deployment schedules and reduces the risks associated with large, infrequent updates. By employing CI/CD pipelines, organizations can ensure their applications are always in a deployable state, react quickly to market changes, and consistently meet customer needs with new features and fixes. The automation of repetitive tasks, such as building, testing, and deploying, not only speeds up the development cycle but also allows developers to focus on creating value rather than maintenance. Moreover, the immediate feedback provided by CI/CD pipelines enables quick resolution of issues, enhancing collaboration and efficiency across teams.
To facilitate the development and deployment of our backend systems and minimize downtime during cutover, we have set up a Custom CI/CD pipeline using Jenkins, an open-source software for automating deployment using pipelines hosted on Digital Ocean's Ubuntu Virtual Machine.

Jenkins will poll GitHub, the Source Change Management platform used for this project every minute to query for changes and commits made to the repository. The pipeline written will deploy the backend and NLP solution when changes have been made to

their respective directories. When such changes are detected, Jenkins will trigger their respective build stage to build the Docker Container using a Dockerfile, a file command to specify the building steps of the system and deploy the new changes (See Figure 15).

```
pipeline {
    agent any
    tools{
        maven 'maven_3_9_5'
    }
    stages{
        stage('Build Maven'){
            steps{
                checkout([$class: 'GitSCM', branches: [[name: '*/main']], extensions: [[$class: 'GitLFSPull']], userRemoteConfigs: [[credentialsId: '68abdbaa-2131-4b53-abcc-1
                sh 'cd Backend && mvn clean install'
            }
        }
        stage('Build docker image'){
            when { changeset "Backend/*"}
            steps{
                script{
                    sh 'cd Backend && docker build -t critiq/backend .'
                    sh "docker stop backend || true"
                    sh "docker rm backend || true"
                    sh 'docker rmi $(docker images -f "dangling=true" -q)'
                    sh 'docker run -d --network="host" --name backend critiq/backend'
                }
            }
        }
        stage('Build NLP'){
            when { changeset "NLP/*"}
            steps {
                script{
                    sh 'cd NLP && docker build -t nlp/backend . --no-cache'
                    sh "docker stop nlp || true"
                    sh "docker rm nlp || true"
                    sh 'docker rmi $(docker images -f "dangling=true" -q) || true'
                    sh 'docker run -d --network="host" --restart unless-stopped --name nlp nlp/backend'
                }
            }
        }
    }
}
```

*Figure 15 Jenkinsfile pipeline written for deploying the Backend Server and NLP Server with the use of docker and dockerfiles*

### 3.3.7 Message Queue

RabbitMQ will be used to support inter-process communication and maintain a durable message queue for high fault tolerance and asynchronous communication. Message Queue (See Figure 16) will be used to support real-time and batch-processing ML features, enhancing system performance and reliability in case of system failure.

RabbitMQ offers various benefits including Durability, Reliability, and Scalability.

- Durability: RabbitMQ can persist messages to disk, ensuring that they are not lost in case of a failure or a restart. This also allows for message recovery and replay.

- Reliability: RabbitMQ provides various features to ensure the delivery and processing of messages, such as acknowledgments, confirmations, dead letter queues, and transactions. These features help to avoid message loss, duplication, or corruption.

- Scalability: RabbitMQ can handle high volumes of messages and concurrent connections, as well as distribute the load across multiple nodes in a cluster. Horizontal scaling can be done easily by deploying more Python Programs without any code modification to handle a larger amount of machine learning tasks.

Compared to other popular Message Queue solutions, including Apache Kafka and Amazon SQS, RabbitMQ offers two distinct advantages: Flexible Routing, and intuitive Management User Interface.

- Flexible Routing: RabbitMQ supports different types of exchanges and bindings, which allow for flexible and dynamic routing of messages based on various criteria, such as topic, header, or direct, and allow for dynamic and flexible routing of messages.

- Intuitive Management User Interface: RabbitMQ provides a web-based user interface that allows for easy monitoring and management of the broker, such as viewing queues, exchanges, bindings, messages, connections, channels, and statistics. The user interface also allows for performing common operations, such as creating, deleting, purging, or publishing messages.

The official Spring and Python RabbitMQ adaptors will be used to connect the Spring Boot Backend and the Python NLP program to the message queue server.

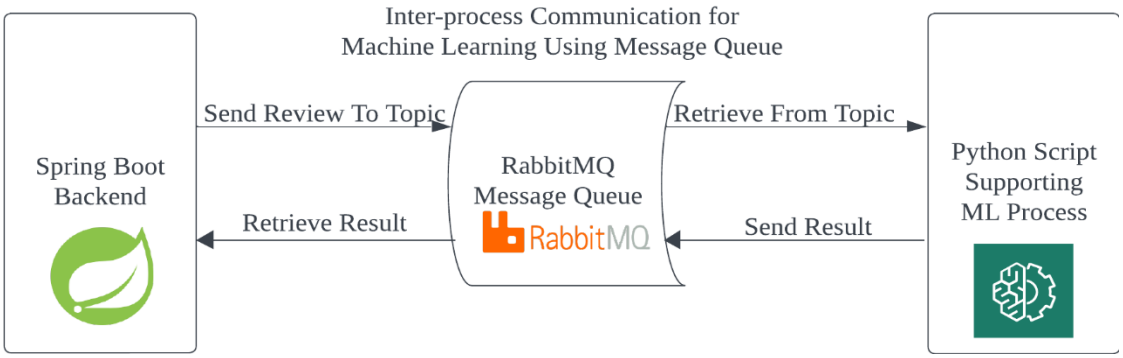



*Figure 16 Message Queue Structure for Supporting Inter-process Machine Learning Application*

The connections to the message queue server and the data stored in the queues can be accessed using a web-based management panel provided for debugging and analytic purposes. The first two connections are the Spring Boot Server and NLP Server Connection to the queues with the last being a local connection (See Figure 17).

| Overview | | | Details | | | Network | |
|---|---|---|---|---|---|---|---|
| **Name** | **User name** | **State** | **SSL / TLS** | **Protocol** | **Channels** | **From client** | **To client** |
| **109.[redacted]:49170**<br>rabbitConnectionFactory#3555d804:0 | FYP | running | ○ | AMQP 0-9-1 | 1 | 0 B/s | 0 B/s |
| **109.[redacted]:60836**<br>? | FYP | running | ○ | AMQP 0-9-1 | 1 | 2 B/s | 2 B/s |
| **42.[redacted]:57810**<br>rabbitConnectionFactory#6c8e40fc:0 | FYP | running | ○ | AMQP 0-9-1 | 2 | 0 B/s | 0 B/s |

*Figure 17 RabbitMQ Management Panel showing all the queue connections.*

### 3.3.8 Hosting

The backend services will be hosted on two different cloud providers, Digital Ocean, and Contabo.

Digital Ocean will host the MySQL Database, RabbitMQ Message Queue Server, and S3-compatible storage Bucket. Digital Ocean provides one-click setup and monitoring for these services on their control panel without needing to create separate virtual machines.

Contabo will only host the Virtual Machine running Ubuntu LTS 21. Virtual Machine hosted by Contabo provides high performance at a low cost of entry with high-bandwidth networking included.

All backend services will be deployed in Singapore, a region that is supported by both Digital Ocean and Contabo. By consolidating all our services in Singapore, we can reduce the latency and network traffic time between API calls and network requests and enhance the overall performance of the entire backend system. Complex API calls such as Advanced Searching or Analytics involve multiple database queries, which would be adversely affected by increasing the physical distance between the database and the Spring Boot server from sub-1 second to over 2.5 seconds.

### 3.3.9 Monitoring

To ensure stable performance and reliability, we use Prometheus and Grafana, two open-source software for monitoring and analytics, to monitor our Spring Boot backend application on our Virtual Machine. Prometheus collects and stores time-series data from the application actuator endpoints, and Grafana visualizes and analyzes the data in dashboards and panels.

A Grafana dashboard will be created to monitor essential information about the Spring Boot Application, including Application Uptime, CPU utilization, Application Load (see Figure 18), and Database Connection Size (See Figure 19). If the application goes down for an extended period, a custom notification will be sent through a webhook.

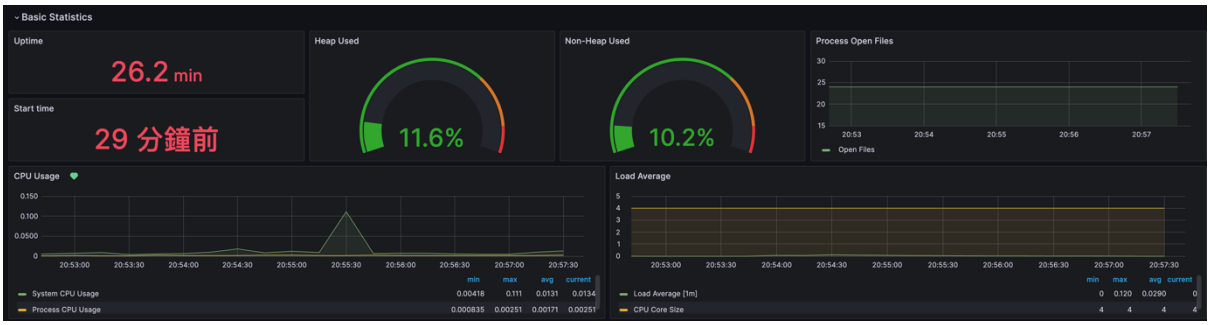



*Figure 18 Grafana Dashboard displaying uptime, CPU, and Memory Utilization by the Spring Boot Application.*

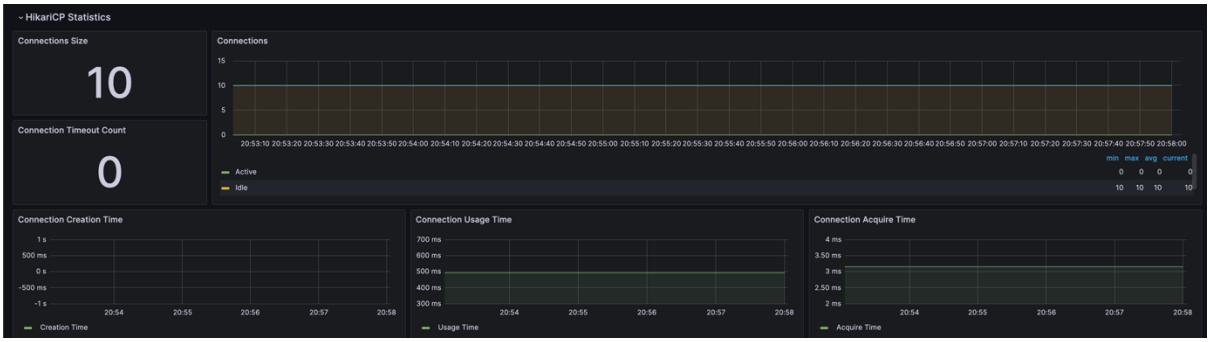



*Figure 19 Grafana Dashboard shows the database connection pool size, maintaining a stable connection to the database.*

# 4 Results

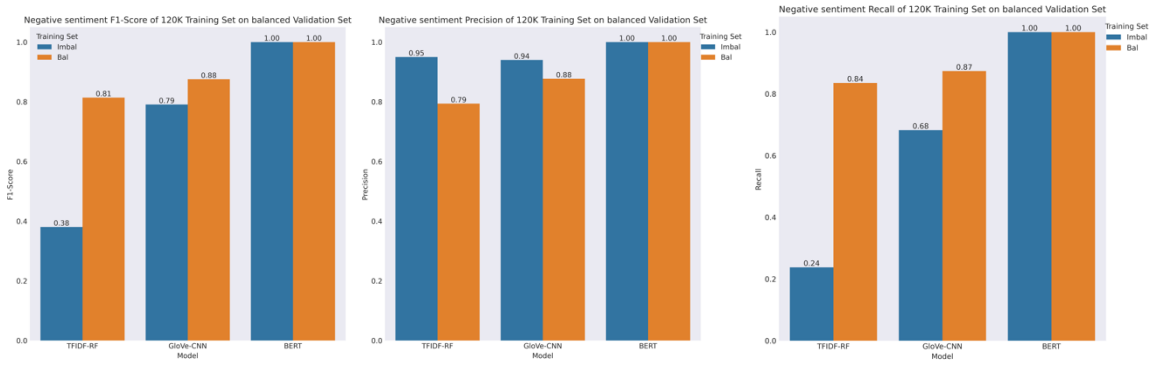
This section discusses the three main results of the project development: Machine Learning and Natural Language Processing (Section 4.1), Frontend Web Applications (Section 4.2), and Backend Technologies (Section 4.3).

## 4.1 Sentiment Analysis

The evaluation was conducted on both balanced and imbalanced validation sets to address the three research questions, thus selecting the best model for deploying on the VM.

**RQ1: Does an imbalanced training dataset hamper model performance?**

Results of all models trained with 120K balanced and imbalanced datasets were presented in Figure 20.



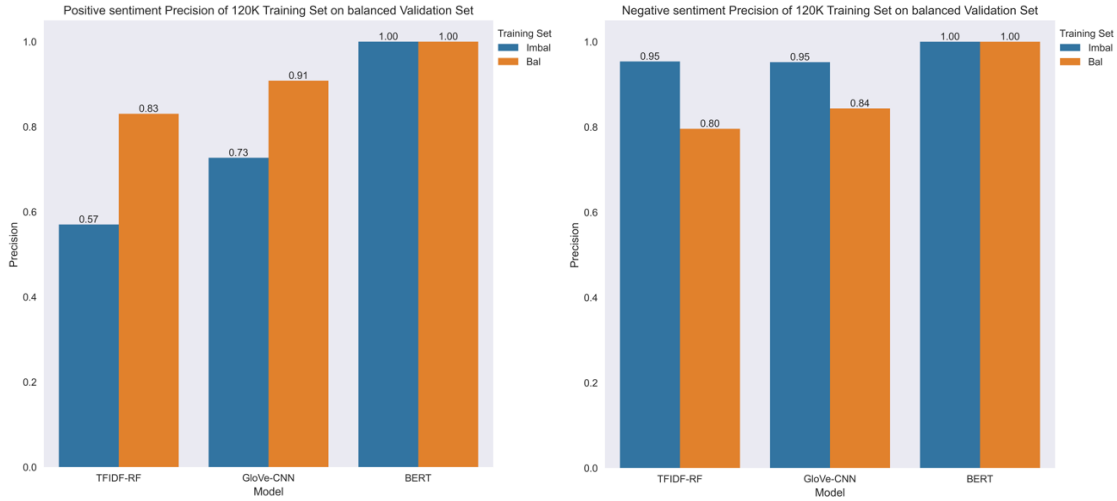

(a) (b) (c)

*Figure 20: Results of all models trained with 120K balanced dataset. (a): Weighted Average F1-score on all models. (b): Weighted Average Precision on all models. (c): Weighted Average Recall on all models.*

It was observed that TFIDF-RF and GloVe-CNN received a lower weighted average F1-score on the balanced validation set, with TFIDF-RF models receiving the biggest difference in the score between training with a balanced dataset and an imbalanced dataset. Since the weighted average F1-score considered both weighted average recall and precision, the difference was surmised to be a drop in recall and/or precision. Considering the weighted average recall of models trained with the 120K dataset, both TFIDF-RF and GloVe-CNN models trained with an imbalanced dataset received lower recall than those with the balanced dataset. The same result was also noted in precision.

To further dig into the cause of the difference in weighted average precision and recall, a comparison was drawn on these metrics in both positive and negative sentiment classes. Regarding weighted average precision, although achieved higher precision in classifying negative sentiment, models trained with an imbalanced dataset fell short in correctly classifying positive sentiment samples (See Figure 21). Models trained with a balanced dataset achieved more all-rounded performance in precision, resulting in higher weighted average precision (See Figure 20(b)). It is surmised that models trained with imbalanced

datasets implicitly learned the distribution of the training dataset, and then made more positive guesses to reviews, leading to lower precision in positive sentiment, and higher precision in negative sentiment. While models trained with balanced datasets attained balanced performance in both sentiment classes, resulting in higher weighted average precision in balanced datasets. Regarding weighted average recall, although achieved higher recall in positive sentiment, models trained with an imbalanced dataset fell short in recalling negative sentiment samples (See Figure 21). While models trained with a balanced dataset achieved more all-rounded performance in recall (see Figure 20(c)), resulting in a higher weighted average recall. The observation further supported our earlier conjecture, leading to a much lower recall of negative reviews, and achieving a nearly perfect recall of positive reviews.




(a) (b)

*Figure 21: Precision of both sentiments on balanced validation set by models trained with 120K imbalanced and balanced datasets. (a): Positive. (b): Negative.*

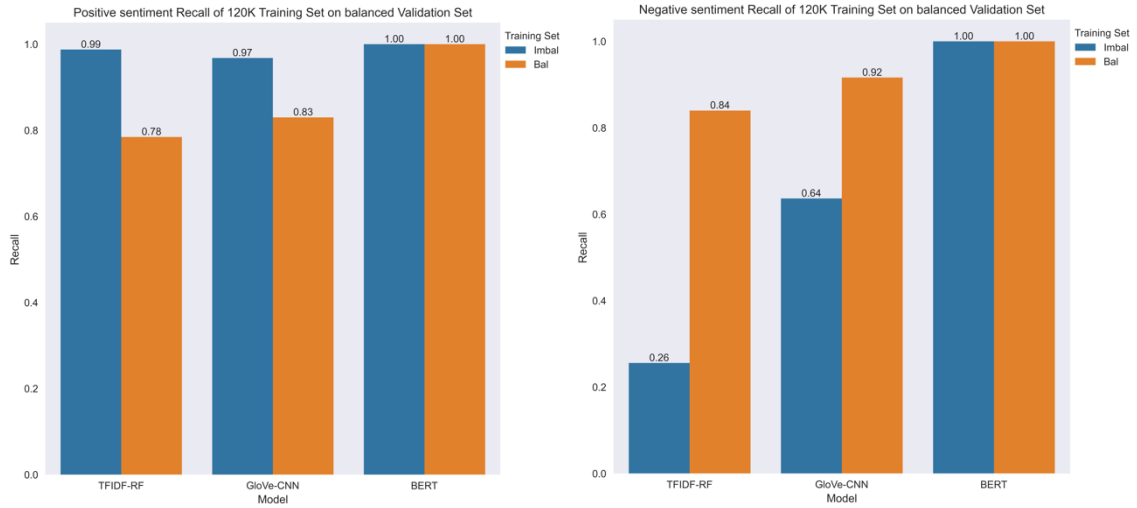
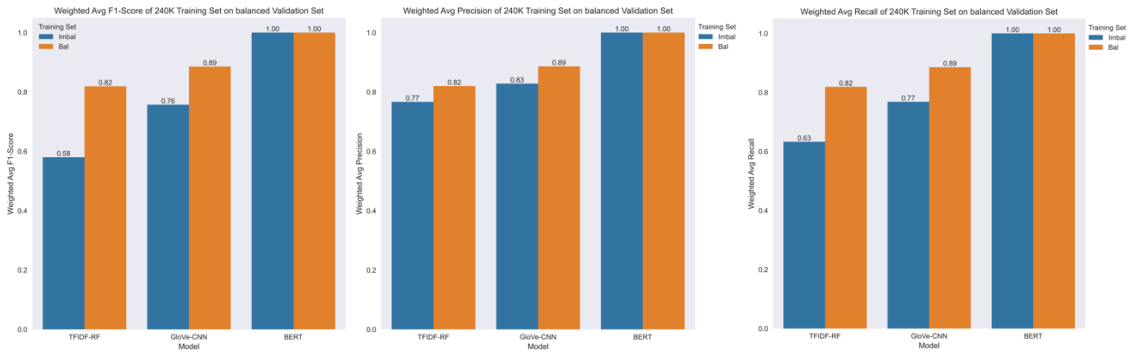


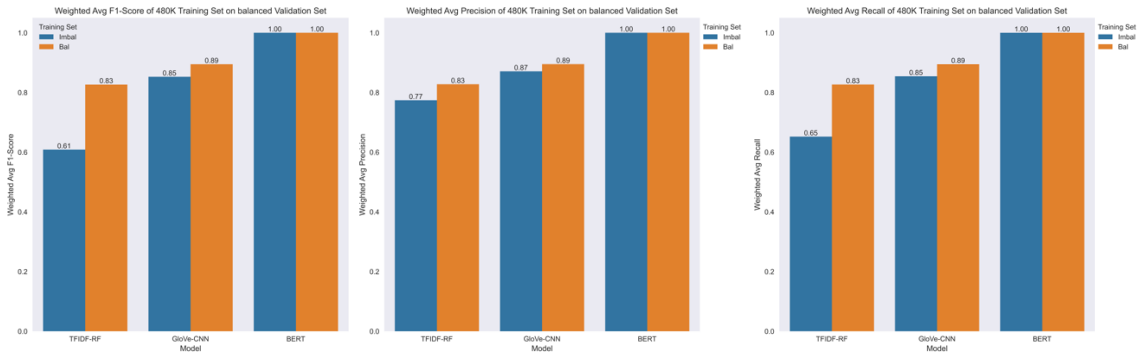

*Figure 22: Recall of both sentiments on balanced validation set by models trained with 120K imbalanced and balanced datasets. (a): Positive. (b): Negative.*

Referring to Figure 23 and Figure 24: Results of all models trained with 480K balanced dataset. (a): Weighted Average F1-score on all models. (b): Weighted Average Precision on all models. (c): Weighted Average Recall on all models., the same conclusion that models trained with an imbalanced dataset underperformed those trained with a balanced dataset was observed in training with both 240K and 480K training datasets. The consistent observation suggested that training with a balanced dataset was preferred, as it yielded higher and more balanced performance.




(a) (b) (c)

*Figure 23: Results of all models trained with 240K balanced dataset. (a): Weighted Average F1-score on all models. (b): Weighted Average Precision on all models. (c): Weighted Average Recall on all models.*

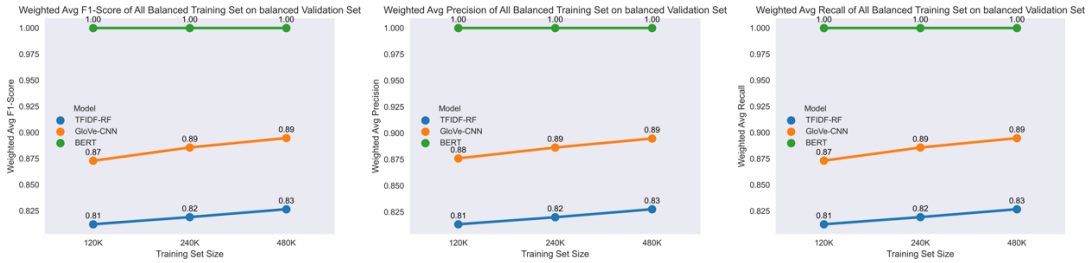



(a) (b) (c)

*Figure 24: Results of all models trained with 480K balanced dataset. (a): Weighted Average F1-score on all models. (b): Weighted Average Precision on all models. (c): Weighted Average Recall on all models.*

### RQ2: What is the relationship between dataset size and performance?

Referring to Figure 25 (a), among the models trained with a balanced training set with different sizes, the Weighted Average F1-score of TFIDF-RF and GloVe-CNN increased as the size of the training set increased. The percentage increase was between 0.84% and 1.46% (See Table 1 (Up)). Further breakdown of the Weighted Average F1-score showed that both Weighted Average Precision and Recall increased as the size of the training set increased. The percentage increase in Weighted Average Precision was between 0.82% and 1.17% (See Table 1 (Bottom left)), while that in Weighted Average Recall was between 0.84% and 1.44% (See Table 1 (Bottom right)).



(a) (b) (c)

*Figure 25: Results of all models trained with balanced datasets of all three sizes. (a): Weighted Average F1-score on all models. (b): Weighted Average Precision on all models. (c): Weighted Average Recall on all models*

| Model/Size | TFIDF-RF | GloVe-CNN | BERT |
|---|---|---|---|
| 120K | 0% | 0% | 0% |
| 240K | +0.84% | +1.46% | 0% |
| 480K | +0.93% | +1.02% | 0% |

| Model/Size | TFIDF-RF | GloVe-CNN | BERT |
|---|---|---|---|
| 120K | 0% | 0% | 0% |
| 240K | +0.84% | +1.46% | 0% |
| 480K | +0.93% | +1.02% | 0% |

| Model/Size | TFIDF-RF | GloVe-CNN | BERT |
|---|---|---|---|
| 120K | 0% | 0% | 0% |
| 240K | +0.84% | +1.46% | 0% |
| 480K | +0.93% | +1.02% | 0% |

*Table 2: Percentage change of all models trained with balanced datasets with different sizes. Up: Weighted Average F1-score. Bottom left: Weighted Average Precision. Bottom right: Weighted Average Recall.*

A similar result can be observed in models trained with imbalanced datasets of all three sizes. The Weighted Average F1-score also increased as the size of the imbalanced dataset increased, except for model GloVe-CNN, which suffered from a drop in performance in the 240K imbalanced dataset (See Figure 26). Considering only positive percentage changes, the percentage increase of the Weighted Average F1-score ranged between 3.05% and 12.61% (See Table 2 (Up)). Further breakdown of the Weighted Average F1-score showed that both Weighted Average Precision and Recall increased as the size of the training set increased. Considering only positive percentage changes, the percentage increase in Weighted Average Precision was between 0.61% and 5.15% (See Table 2 (Bottom left)), while that in Weighted Average Recall was between 1.81% and 11.20% (See Table 2 (Bottom right)).

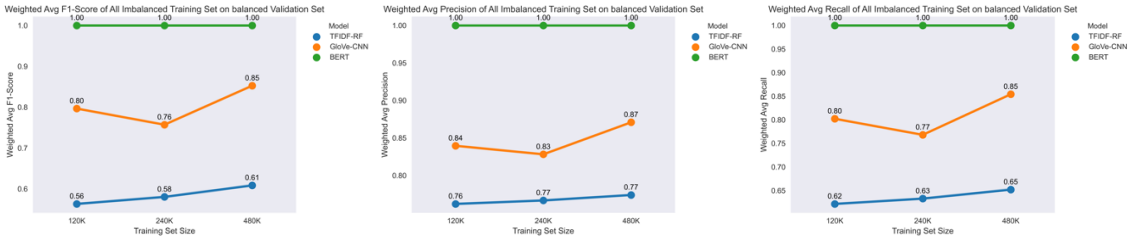



*Figure 26: Results of all models trained with imbalanced datasets of all three sizes. (a): Weighted Average F1-score on all models. (b): Weighted Average Precision on all models. (c): Weighted Average Recall on all models*

| Model/Size | TFIDF-RF | GloVe-CNN | BERT |
|---|---|---|---|
| 120K | 0% | 0% | 0% |
| 240K | +3.05% | -4.96% | 0% |
| 480K | +4.85% | +12.61% | 0% |

| Model/Size | TFIDF-RF | GloVe-CNN | BERT |
|---|---|---|---|
| 120K | 0% | 0% | 0% |
| 240K | +0.61% | -1.35% | 0% |
| 480K | +0.95% | +5.15% | 0% |

| Model/Size | TFIDF-RF | GloVe-CNN | BERT |
|---|---|---|---|
| 120K | 0% | 0% | 0% |
| 240K | +1.82% | -4.24% | 0% |
| 480K | +3.02% | +11.20% | 0% |

*Table 3: Percentage change of all models trained with imbalanced datasets with different sizes. Up: Weighted Average F1-score. Bottom left: Weighted Average Precision. Bottom right: Weighted Average Recall.*

**RQ3: What is the best model with little hyperparameter selection?**

| Model | Balanced training set | | | Imbalanced training set | | |
|---|---|---|---|---|---|---|
| | 120K | 240K | 480K | 120K | 240K | 480K |
| TFIDF-RF | 0.81 | 0.82 | 0.83 | 0.82 | 0.83 | 0.84 |
| GloVe-CNN | 0.89 | 0.90 | 0.91 | 0.91 | 0.91 | 0.92 |
| BERT | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** |

*Table 4: Weighted Average F1-Score of all models on the imbalanced validation sets.*

| Model | Balanced training set | | | Imbalanced training set | | |
|---|---|---|---|---|---|---|
| | 120K | 240K | 480K | 120K | 240K | 480K |
| TFIDF-RF | 0.81 | 0.81 | 0.82 | 0.55 | 0.57 | 0.59 |
| GloVe-CNN | 0.88 | 0.88 | 0.89 | 0.82 | 0.81 | 0.83 |
| BERT | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** |

*Table 5: Weighted Average F1-score of all models on the balanced validation sets.*

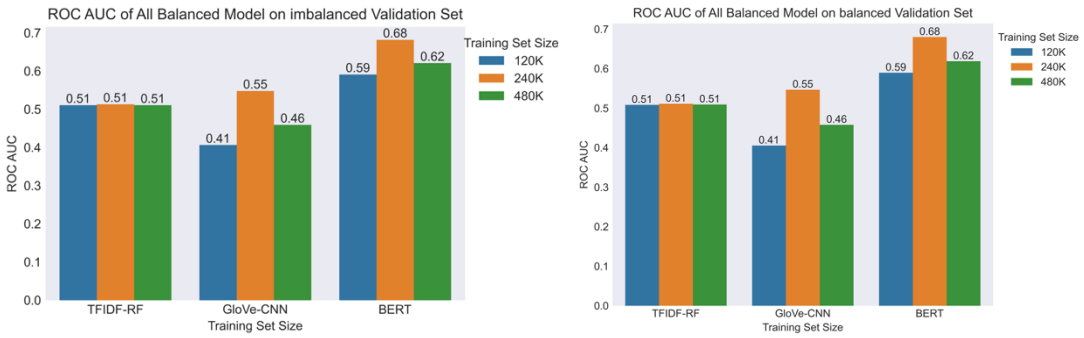
Referring to Table 3, both models trained with an imbalanced dataset and balanced dataset received similar weighted average F1-score on the imbalanced validation set, even the models trained with an imbalanced dataset slightly outperformed those trained with the balanced dataset. However, in Table 4, as mentioned in RQ1, models trained with an imbalanced dataset fell short in performance in a balanced validation set.

Although most games receive chiefly positive reviews, there will be situations in which mixed reviews or more extreme, mostly negative reviews will be received on the game platform. They can occur due to the game's poor quality, sluggish gameplay, or frequent bugs, resulting in more than usual negative reviews. Examples are The Last of Us 2, the initial release of No Man's Sky, and Lord of Rings: Gollum. Moreover, some of the games were under early access reviews, in which a demo of the game was released on the platform for eager players to test and provide feedback although the game was under development process. Since the game was unpolished and incomplete, it was expected for the developers to receive a mixed review. Therefore, considering the performance of models in both balanced and imbalanced validation datasets, models trained with balanced datasets were preferred. Referring to Table 3 and Table 4, the best performant model was BERT, achieving a perfect weighted average F1-score in both imbalanced and balanced validation datasets.

Since there was a tie in the weighted average F1-score of all BERT models, a different metric is required to select the best performant model with respect to the size of the training dataset. Receiver Operating Characteristic – Area Under the Curve (ROC-AUC) was

chosen to be the metric as it measures the performance of a model at different classification thresholds. A prediction from a model can be treated as a probabilistic prediction, with each class containing a value [0, 1], and the sum of them equal to 1. With the information, the True Positive Rate (TPR) against the False Positive Rate (FPR) at different prediction thresholds can be plotted, creating the ROC curve. To find out the ROC-AUC value, the area under the ROC curve is calculated. A ROC-AUC value = 0.5 represents a random classifier, and a ROC-AUC value = 1.0 represents a perfect classifier. Classifiers with ROC-AUC value < 0.5 are considered worse than a random classifier, and vice versa. The higher the ROC-AUC value, the more performant the classifier is.

Referring to Figure 27, BERT fine-tuned with 240K balanced dataset scored the highest ROC-AUC among all three BERT models in the imbalanced validation set, achieving a 0.68 in ROC-AUC. A similar result on a balanced validation set was also observed. Therefore, the BERT model fine-tuned on a 240K balanced training set was the best performant model. Its ROC curves on both imbalanced and balanced validation sets are displayed in Figure 28.




(a) (b)

*Figure 27: ROC-AUC of all models trained with the balanced dataset. (a): on the imbalanced validation set. (b): on the balanced validation set.*

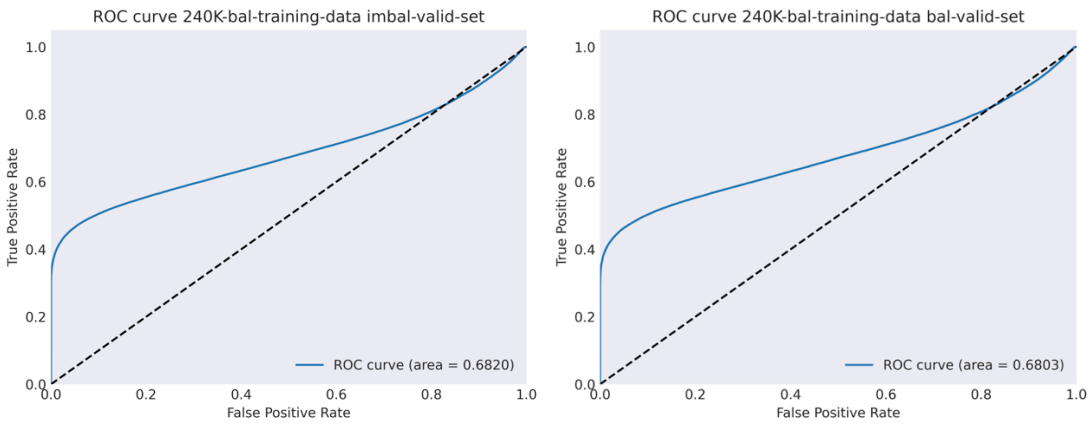
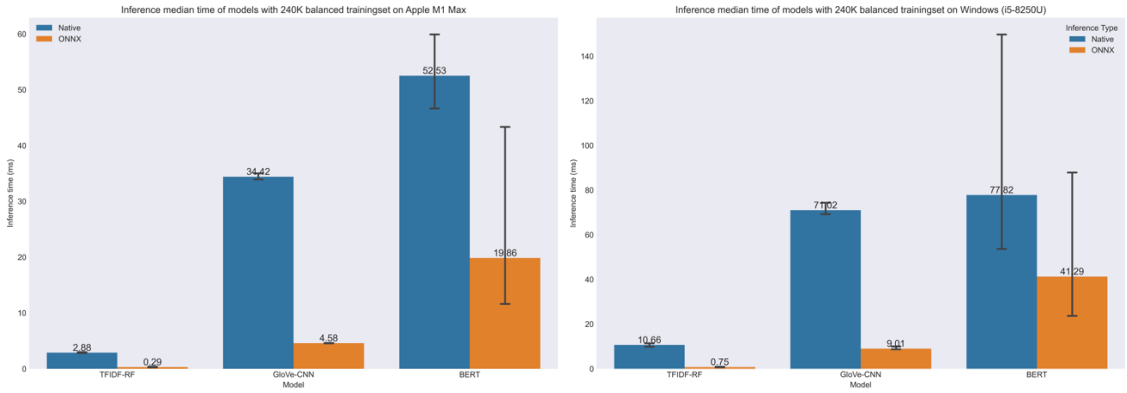



*Figure 28: ROC curve of BERT fine-tuned on 240K balanced training set. (a): on the imbalanced validation set. (b): on the balanced validation set.*

Regarding the required inference time for one review, it was obvious that speedup was recorded for all three models on both machines. The median inference time, lower and upper quartile were plotted in Figure 29. Among the three models, TFIDF-RF recorded the largest speedup on both machines, followed by GloVe-CNN, and lastly BERT (See Table 6). The large variation in inferencing times on BERT was attributed to its higher complexity compared to the other two models.




(a) (b)

*Figure 29: Median inference time of original and ONNX model on different machines. (a): Machine 1 (Windows i5-8250U). (b): Machine 2 (Apple M1 Max)*

| | Machine 1 (i5-8250U) | Machine 2 (M1 Max) |
|---|---|---|
| TFIDF-RF | 20.23 | 8.42 |
| GloVe-CNN | 7.90 | 7.49 |
| BERT | 1.81 | 2.71 |

*Table 6: Median Speedup of inference time of both machines.*

Therefore, considering the conclusion of three research questions and results on inference time evaluation, BERT finetuned on a 240K balanced training set, converted to ONNX format, was selected for deployment on the VM.

### 4.2 Topic Modeling and Keyword Extraction

This section discusses the implementation and application of Topic Modeling and Keyword Extractions, using NLP, to automate the analysis of game reviews on our platform.

Topic Modeling is an unsupervised machine learning technique that helps to discover the hidden thematic structure in a large corpus of text. It identifies topics present in a collection of documents by clustering similar words together. For instance, in-game review analysis, and topic modeling can group reviews that discuss graphics, gameplay, sound design, or story, helping to understand common themes across numerous reviews.

Keyword Extraction, on the other hand, involves automatically identifying the most relevant and significant terms from a single document or a collection of documents. This process helps to quickly understand the main points by extracting words or phrases that capture the essence of the text. In-game review analysis, keyword extraction can highlight the most talked-about features or issues in a game, such as “stunning visuals,” “difficult controls,” or “engaging story” paraphrased or extracted from the review.

As discussed, 3 datasets will be created, with all reviews set being around 4 Million, the Action genre set being 1.31 Million, and the Indie genre set being 0.72 Million reviews.

Our approach combines the two techniques in analyzing a review, providing detailed information interpreted by the content of the review, and creating size information for the users to quickly understand the review, using the steps below (See Figure 30 and Figure 31).

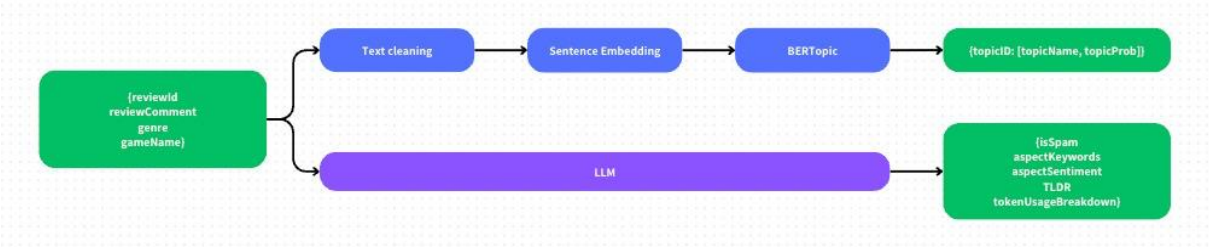



*Figure 30 Technologies for Keyword Extraction and Topic Modeling on Review Creation*

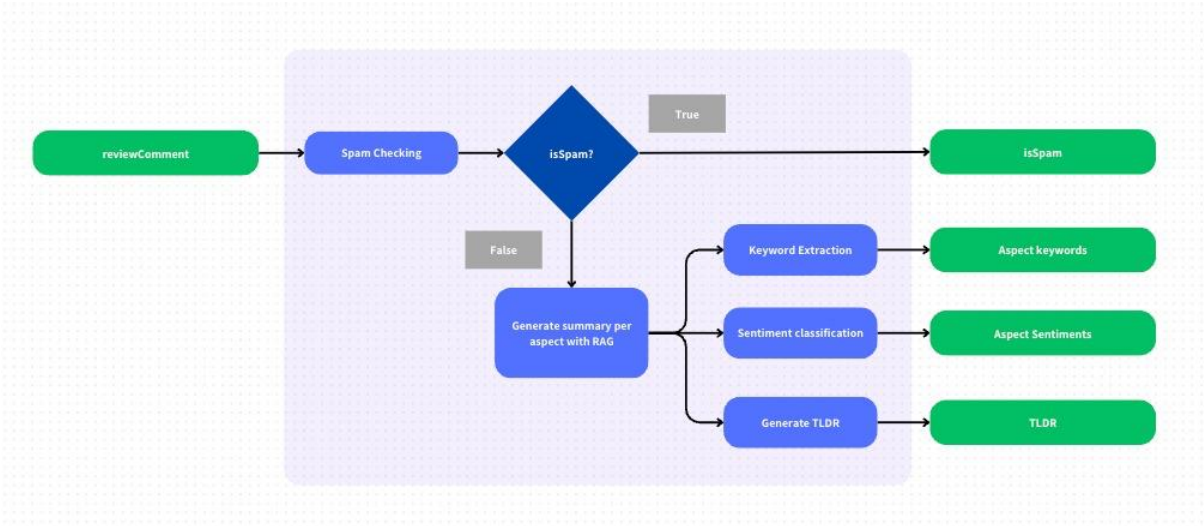



*Figure 31 Flowchart for Keyword Extraction, Topic Modeling, and Summarization*

When a review is created, a message containing the review ID, review comment, game genre, and game name will be sent to a Queue which will be read by our Python Program. When the message is read, it will trigger 2 main functions, one for Topic Modeling and one for Keyword Extraction.

The selection of the Topic Modeling model from the 3 models listed is based on **Quantitative and Qualitative evaluation**:

- **Quantitative**

  Quantitative aims to select the 2 best performing models for further qualitative analysis. Two measurements will be compared

  1. Topic Coherence

     Normalized Pointwise Mutual Information (NPMI) is a common evaluation metric for assessing the quality of topics discovered by measuring the association strength between words within a topic by considering word co-occurrence and individual frequencies. NPMI ranges from -1 (no association) to 1 (strongly related within 1 topic). By measuring the Topic Coherence metrics, the “supportiveness” of a topic by the reference corpus can be measured and evaluated (See Figure 32).

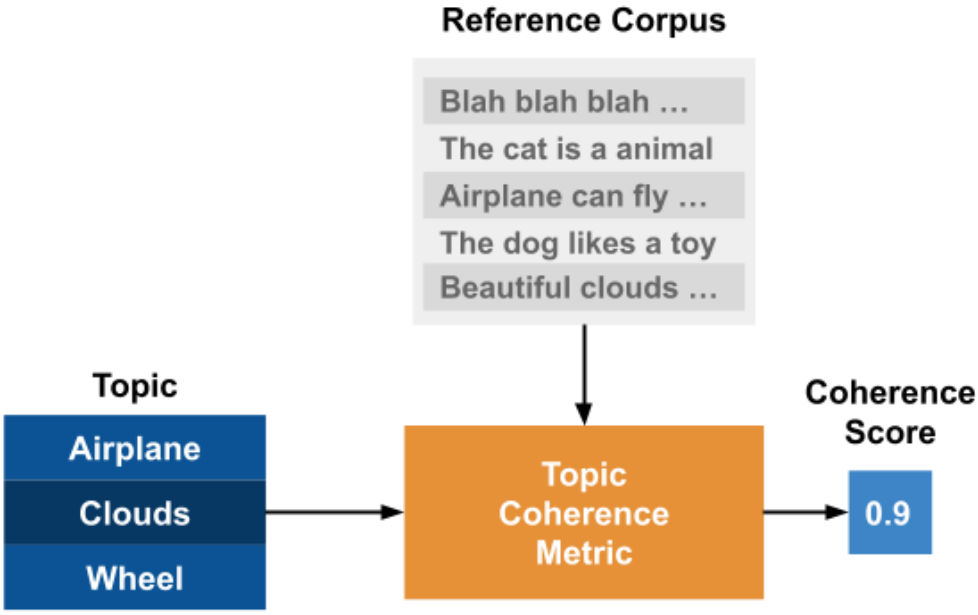



*Figure 32 Topic Coherence Score example of a Topic consisting of 3 words against a Reference Corpus.* https://towardsdatascience.com/understanding-topic-coherence-measures-4aa41339634c

The GenSim python library provides an implementation of a class that imports 4 of the most coherent models: u_mass, c_v, c_uci, and c_npmi. In our case, we will focus on c_npmi.

C_NPMI takes into consideration the 4 following factors:

- Segmentation: c_npmi makes uses of the S-1-1 method, which compute the confirmation over pairs of words instead of a set (Röder et al., 2015).
- Probability Calculation: This approach employs the Psw(10) technique, where probabilities are determined using a moving window of 10 units in length that traverses the text.
- Confirmation Measure: The confirmation metric for each duo is determined by the Normalized Pointwise Mutual Information (NPMI), see the equation below.

$$\mathrm{NPMI}(x,y) = \frac{\log\left(\frac{P(x,y)}{P(x)P(y)}\right)}{-\log\big(P(x,y)\big)}$$

- Aggregation: The overall coherence score is calculated as the average of all the confirmation measures.

The graph below shows the C_NPMI score for each of the 3 models (See Figure 33). BERTopic and CTM include a split measurement because of the 384 tokens context size limitation of SBERT and split the text into chunks of 384 tokens to

ensure the models can still learn the different topics within the same review and generate topics individually.

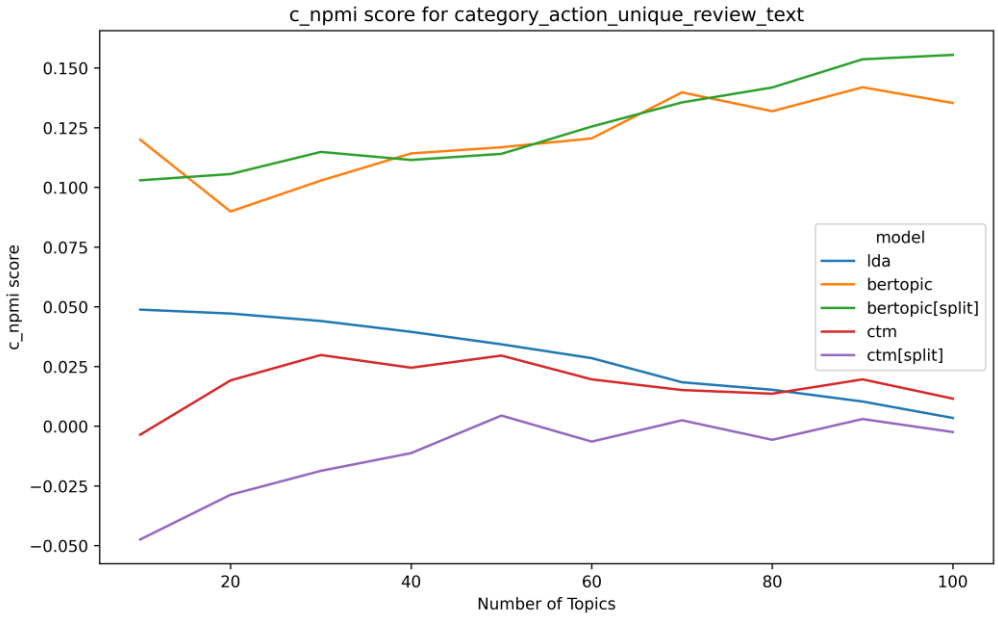



*Figure 33 C_NPMI Measurement of 3 Topic Modeling Models and their performance*

As observed, both BERTopic and BERTopic (Split) outperform LDA and CTM and LDA performs slightly better than CTM and CTM(Split), thus qualitative evaluation will only be performed for BERTopic and LDA.

2. Topic Diversity

The Inverted Rank-Biased Overlap (Inverted-RBO) is a specialized metric designed to quantify the level of disjointedness between topics. This metric places a higher weight on the rankings of words based on their prominence within the top-K words of each topic. As observed (See Figure 34), it clearly illustrates that both LDA and BERTopic exhibit superior performance compared to CTM and CTM(Split). This performance gap becomes increasingly pronounced as the number of topics under consideration expands, highlighting the robustness of LDA and BERTopic in managing larger topic sets..

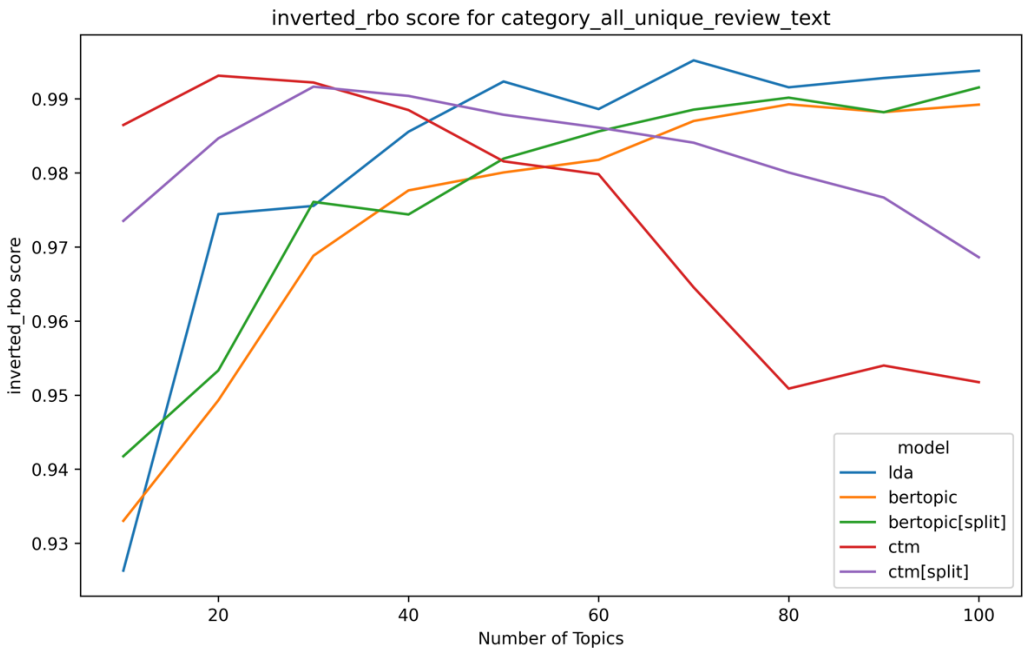



*Figure 34 Inverted RBO Measurement of 3 Topic Modeling Models and their performance*

- **Qualitative**

  Qualitative evaluation requires manual inspection of the result topics to determine its accuracy and effectiveness and the following techniques were used.

  1. **Visualization with pyLDAvis and Distance Map**

     pyLDAvis is a Python library used that provides interactive visualization and interpretation of topic models generated from a fitted LDA model (See Figure 35). The web UI provided by pyLDAvis allows for interactive exploration of the topics found by LDA and visualizes the characters and their relevance to the topics (See Figure 36 BERTopic's provided visualization interface similar to LDAvis, displaying topic overlap and size, quickly understanding the effectiveness of the Topic Modeling model.

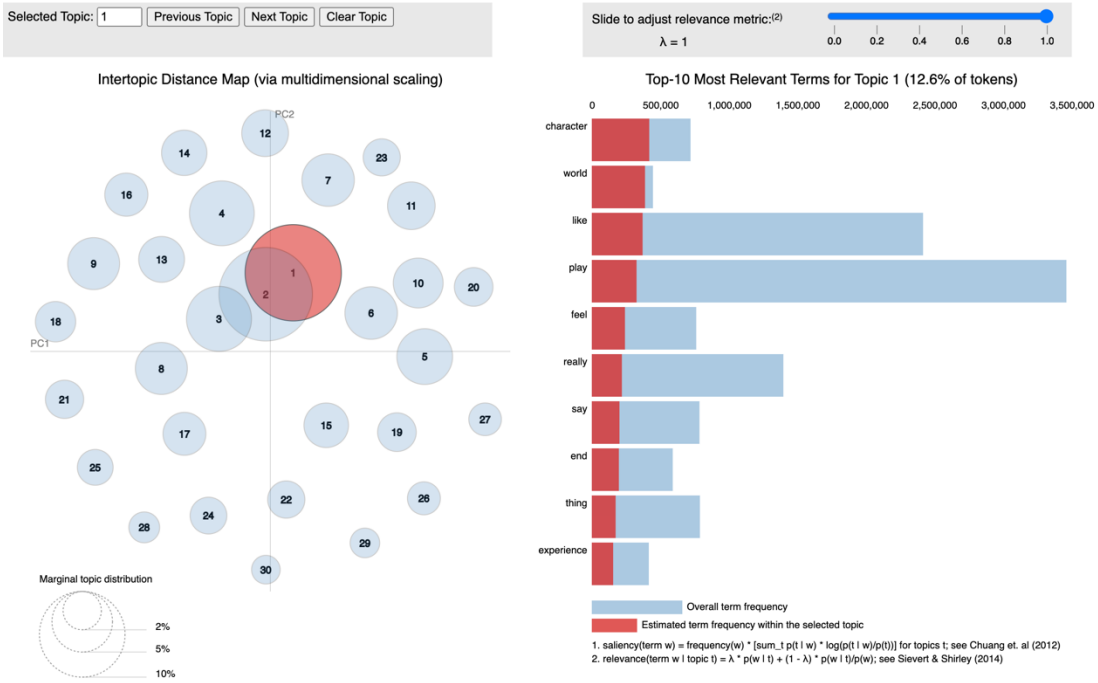



*Figure 35 pyLDAvis web UI showing the most common topic its top keywords and the distance map*

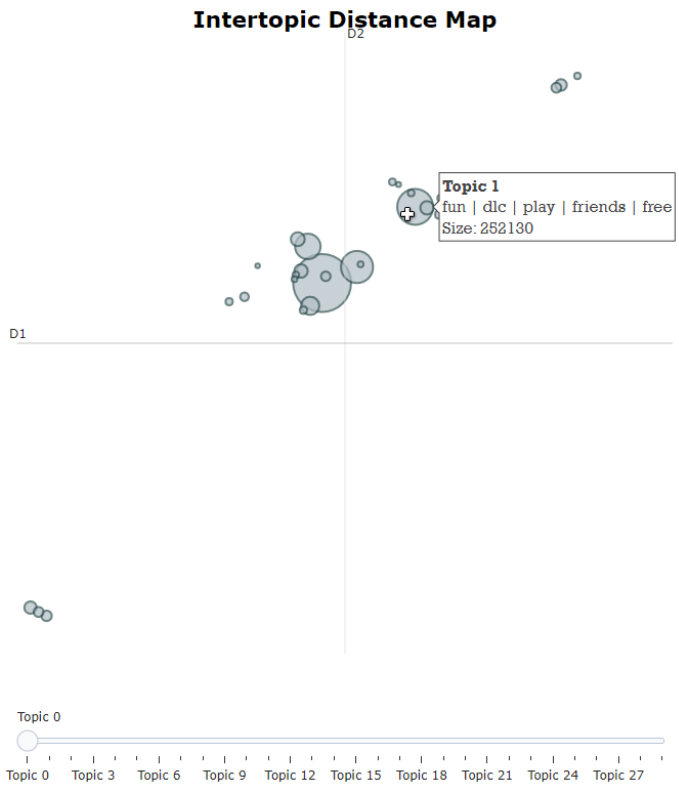



*Figure 36 BERTopic's provided visualization interface similar to LDAvis, displaying topic overlap and size*

Below are two example topic models generated with both LDA and BERTopic (See Figure 37 and Figure 38).

| | | |
|---|---|---|
| | | |
| | 'work', 'fix', 'play', 'crash' 'window' | 'crashes' 'windows' 'crashing' 'crash' 'fix', 'work', 'play', |
| | | |
| | | |

*Figure 37 Cross-model comparison on a similar topic (crashes and bugs)*

| | | |
|---|---|---|
| | | |
| | 'puzzle' 'horror' | 'puzzles', 'puzzle' 'horror' 'scary' |
| | | |
| | | |

*Figure 38 Cros- model comparison on a similar topic (horror puzzle game)*

2. Top 10 Keywords of Each Topic

   The two examples show that BERTopic is more capable of extracting more relevant keywords to the topics than LDA from the large corpus of review text.

3. Representative Texts

   As observed, the top representative document/text from BERTopic is far superior in terms of quality and distinctiveness. LDA tends to favor reviews with many repetitive words, for example, “Puzzles” and “Crash”, because of lemmatization and stemming. BERTopic focuses on the contextual meaning of the sentences and the word associations, making it less likely to identify repetitive words in the representative documents.

4. Topic Name generated by LLM

   Because of the contextual embedding with BERT, the trained model can be applied to new games sharing the same genre/category and create a descriptive and meaningful topic name that can be easily understood by users. The following figures show the generic topic name based on the action genre’s reviews and LLM-generated topic names for the Action-RPG game “Starfield” (See Figure 39,Figure 40).

```
{
  "-1": "Fun Game",
  "0": "Fun Adventure Game",
  "1": "PC Port Issues",
  "2": "Good Shooting Game",
  "3": "Fun Short Game",
  "4": "Learn Russian",
  "5": "Sonic Revival",
  "6": "Cake is lie",
  "7": "Classic Game",
  "8": "Virtual hat Simulator",
  "9": "Whiskey Pass"
}
```

*Figure 39 Topics for Action Games generated by Trained Topic Modeling model*

```
{
  "-1": "Disappointing experience with too much repetition",
  "0": "Fun Space Exploration",
  "1": "Underwhelming experience with frequent crashes.",
  "2": "10/10",
  "3": "Good Game",
  "4": "Good graphics",
  "5": "Spaceship and fast travel",
  "6": "Lies and Disguise",
  "7": "Masterpiece",
  "8": "Underwhelming Experience with Limited Customization",
  "9": "Fringe Quests and Limited Impact"
}
```

*Figure 40 Transferred Topics for Action Games generated by Trained Topic Modeling model and LLM*

Transfer learning is a powerful machine learning technique where a model developed for a particular task is reused as the starting point for a model on a second task. In the context of Large Language Models (LLMs) and topic modeling, transfer learning can be particularly effective in adapting to new domains or genres with minimal effort. Here's a detailed explanation of how transfer learning was applied to the game "Starfield" (See Figure 40): The transition from "Fun Adventure Game" to "Fun Space Exploration" for "Starfield" exemplifies the efficacy of transfer learning. The process commenced with the LLM analyzing the corpus of reviews for "Starfield," focusing on distinctive features such as "Spaceship" and "Fast Travel." These features, while specific to "Starfield," resonate with the broader concept of adventure within the action genre. Through transfer learning, the LLM leveraged its pre-existing knowledge base—honed through exposure to a diverse array of action games—and applied it to the unique context of "Starfield." The model adeptly recognized the parallels between the general thrill of adventure and the specific allure of space exploration. Consequently, it synthesized a new, more

apt topic name: “Fun Space Exploration.” By drawing on the generalized topics from its training, the LLM could intuitively map these themes onto “Starfield” without necessitating additional adjustments or retraining. The result is a nuanced and accurate representation of the game’s core experience, as reflected in the player reviews. The success of this application demonstrates that transfer learning is not only feasible but also remarkably precise. It affirms the potential of LLMs to evolve and adapt to new content within the same genre, thereby streamlining the integration process for future games and enhancing the overall analytical framework.

In the comparative analysis, BERTopic demonstrated a substantial advantage over both Contextualized Topic Models (CTM) and Latent Dirichlet Allocation (LDA) in terms of both quantitative metrics and qualitative assessments. Consequently, BERTopic was selected as the definitive model for Topic Modeling.

The process begins by loading the Topic Modeling Model that corresponds to the specific game genre. Due to the constraints of the project timeline, models have been trained exclusively for the Action and Indie genres. Each model encompasses a curated set of the ten most pertinent topics, which have been refined through extensive training on a vast corpus of user reviews and subsequent manual curation. The textual content of the reviews is then processed through the Sentence Bert model, specifically the “all-MiniLM-L6-v2” variant, to generate embeddings. These embeddings are subsequently converted into two distinct NumPy arrays: one array encapsulates the identified topics, while the other quantifies the likelihood of each review’s alignment with these topics.

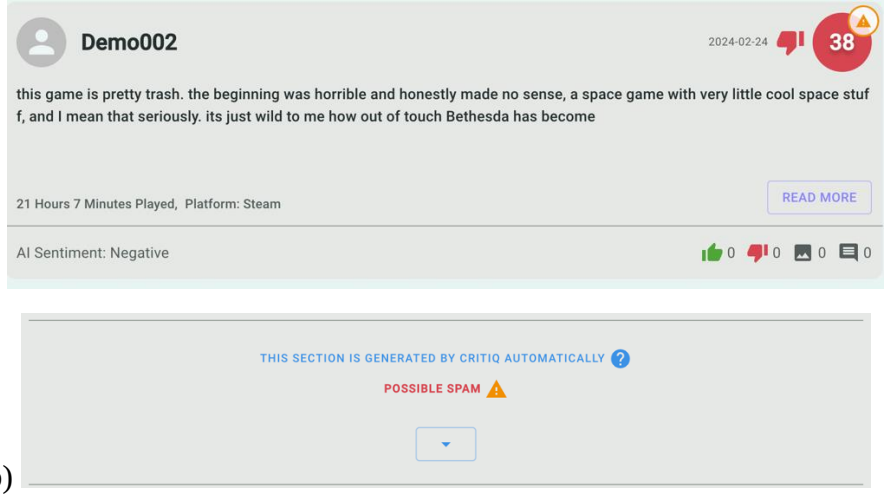
To facilitate effective Topic Modeling, it is imperative to establish a collection of predefined topics. These topics are derived from an analysis of existing professional critic reviews, utilizing the Retrieval Augmented Generation (RAG) technique. RAG functions by indexing the documents, in this instance on a temporary basis, within a vector database known as Chroma DB, which is managed using Docker. Each topic generated through this process is then input into the Large Language Model (LLM) alongside a compilation of reviews pertinent to that topic. This step is crucial for the generation of descriptive topic names, addressing the limitation of BERTopic which provides only numerical identifiers rather than explicit names. Subsequently, when a user submits a review for analysis, the system employs information retrieval and semantic search methodologies to the vectorized

text. This enables the identification of the most closely related or analogous topic within the temporary vector database. The identified topic is then prominently displayed as the Main Topic within the web application's user interface.

Keyword Extraction is employed to distill significant insights from voluminous text data. The process begins with the deployment of custom-designed prompts that serve a dual purpose: firstly, to ascertain the authenticity of a review by determining its spam status, and secondly, to extract pivotal keywords that encapsulate the essence of the review across various gaming aspects.

**LLM Prompt Engineering**

The initial prompt is crafted with precision to evaluate the legitimacy of the review, structured as follows: *"Is the game review spam? Output 'YES' if the review is spam and 'NO' if the review is not a spam. Only output the decision. Do NOT output the reasons. Do NOT output other text."* If a review is tagged as a Spam, a warning icon or label is displayed in the application to signal the user about the authenticity and accuracy of such reviews (See Figure 41).




(a)

(b)

*Figure 41 (a) A warning sign in review cards shown in-game and on all reviews pages; (b) a "Possible Spam" phrase added in the review analysis section*

Subsequently, reviews deemed authentic undergo a meticulous keyword extraction process. This involves a series of specialized prompts, each tailored to one of the ten critical aspects of the gaming experience: Gameplay, Narrative, Accessibility, Sound, Graphics & Art Design, Performance, Bug, Suggestion, Price, and Overall. The objective is to isolate and extract salient keywords that are indicative of the user's sentiments and opinions related to these aspects. To do so, a system prompt and a user prompt are created and multiple prompt engineering techniques are applied to reduce the hallucination and disobeying of the LLM.

The system prompt serves as the initial set of context and guidance for the model and constrains the LLM using a well-defined context to stipulate the LLM's primary task and

thought process, which is “a player reading reviews of a game”, confining the LLM to work within the boundary of game reviews.

| |
|---|
| System: You are a player reading reviews of a game to understand the characteristics of the game. Use the following pieces of context to answer the user's question. |

While the user prompt is the main input query (the review analysis prompt). In particular, the highlighted part showcases the prompting techniques used.

| |
|---|
| User: You are reading reviews of a game to understand the characteristics of the game. Extract the following aspect of the game from the reviews. The aspects are (Lin et al.). For each aspect, output a paragraph with less than 50 words. Then create a JSON with apsects name as key and the paragraph as value.<br>Output the JSON as a single line with no spaces between the key, value pairs. {For example, if the aspects are {aspects_list}, the JSON should be: {”aspect01”:”…”, “aspect02”:”…”, “aspect03”:”….”}}<br>Only output the JSON. Do NOT output other text.<br><br>The reviews are as follows:<br>\'\'\' {summaries} \'\'\'<br>If you don't know the answer, output only \"NA\". Do NOT try to make up an answer. Do NOT output other text.' |

Highlighted Parts:

- Yellow

  The yellow highlighted part is a One-Shot Prompt example in the desired output format where the name of the aspects will replace “aspects01” and the other placeholders during the execution of the prompt. Using a One-Shot prompt example drastically improved the accuracy and tendency of the LLM strictly following the output scheme

that is expected and the same behavior is shared amongst 3 tested small LLMs, including Llama2-7B, Mistral-7B, and Gemma-7B.

- Red

  The red highlighted text further reinforces the output format requirement, reducing the chance of the LLM hallucinating and generating content that does not follow the guideline or is irrelevant.

- Purple

  The actual review text will be dynamically injected into the placeholder as the main context of the review analysis process.

- Blue

  The symbols highlighted in blue serve as delimiters for the context, allowing the LLM to separate the main context from the instruction before and after.

The text below shows an example of the result of the LLM for a review aspect analysis, showing that the LLM can follow the requirements and instructions strictly with the above prompt engineering techniques applied. The result shows no sign of hallucination, being able to reply “NA” for aspects that are not discussed and the output can be easily processed by parsing it as a JSON object, reducing computation need.

"{\"Gameplay\": \"The game features good graphics and smooth gameplay, allowing you to sneak around and use various weapons. Paris is large with many activities like coop missions and random events.\", \"Narrative\": \"NA\", \"Accessibility\": \"A good PC is required to run the game and there are some bugs, but the game is generally playable.\"}"

To further refine the analysis, the Large Language Model (LLM) is prompted to generate a binary sentiment for each aspect (See Figure 42). These sentiments are then systematically highlighted and organized within the web application interface, enhancing the user experience by providing a clear and sorted visualization of the sentiments associated with each gaming aspect. This nuanced approach enables a comprehensive and granular understanding of the collective user feedback, facilitating informed decision-making for

game developers and players (See Figure 43). Furthermore, only aspects that are found in the review will be displayed to reduce clutter and nonessential information (See Figure 44).


```
"sentiment": {
  "Suggestion": "NA",
  "Overall": "Positive",
  "Accessibility": "Positive",
  "Gameplay": "Positive",
  "Narrative": "NA",
  "Bug": "NA",
  "Price": "NA",
  "Sound": "NA",
  "Performance": "Positive",
  "Graphics & Art Design": "Positive"
},
```


*Figure 42 Sentiment generated per aspect for review*

**Key Words:**

Graphics & Art Design:
so much to do, so much to explore, fantastic new adventure

Performance:
many loading screens, unacceptable load times, no land vehicles

Bug:
invisible wall, replicated points of interest, numerous loading screens, lack land vehicles, issues temple designs

Gameplay:
repetitive activities, lack of depth, invisible walls, no land vehicles, numerous loading screens

Accessibility:
numerous loading screens

Overall:
good aspects, overshadowed bugs, lack of depth, repetitive, current gaming trends

*Figure 43 Web Application Displaying Aspects sorted by their sentiments*

**AI Sentiment:** Positive

**Main Topics:** Cyberpunk DLC - Great Expansion

**Key Words:** Overall:
Pretty good demo showcase of the game features!

*Figure 44 Keywords Section only display found aspects*

To accommodate users who seek a concise overview of extensive reviews, a summarization feature is implemented, generating a “Too Long; Didn’t Read” (TL;DR) or summary segment (See Figure 45). This component distills the core themes and opinions of a review into a brief abstract, enabling users to rapidly grasp the fundamental content without delving into the full text. This efficient synthesis of information ensures that even the most comprehensive reviews are accessible and comprehensible at a glance.

| | |
|---|---|
| **Summary:** | Game has wide-ranging but shallow gameplay, amazing graphics, numerous loading screens, lack of land vehicles, and many bugs, overshadowing its good aspects. |

*Figure 45 Web Application Displaying Summary of the Review*

The following Prompting Techniques were used to improve the accuracy outputs:

- **One-Shot Prompting**

  A single prompt or example is provided to the LLM to guide the LLM's response by leveraging the model's pre-trained knowledge and general understanding to perform tasks without specific prior training. In particular, we want to enforce a JSON output format for better data processing of the result from the LLM.

- **Use of Delimiters**

  Delimiters help structure prompts by separating different instructions or components. Special tokens like [SEP] or [EOS] were used to indicate the end of one instruction and the start of another. Properly placed delimiters improve the LLM's comprehension and guide it toward the desired output.

- **Role Prompting**

  Role prompting assigns specific roles or perspectives to the LLM. Framing the prompt as if the model were a character or expert, encourages context-aware responses. It is useful for generating content from different viewpoints or simulating conversations, for example, prompting the LLM's role to be a "Game Reviewer" reading the reviews of a game and its statistics, allowing the LLM to be more in-character, reducing hallucination and making the result more focused on analyzing the review.

### 4.3 Game Aggregated Review

To generate a TL;DR (Too Long; Didn't Read) summary, a game needs to have a sufficient number of reviews and professional critiques. This summary aims to provide a concise yet informative overview of the game that does not include superficial measurements such as review scores and mainly considers the ideas expressed in the reviews. There are 5 main steps in generating a game-aggregated review (See Figure 46).

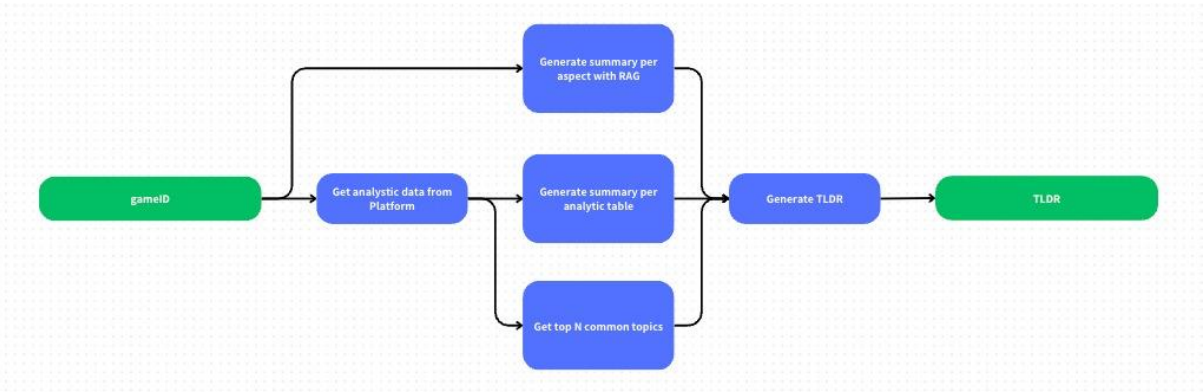



*Figure 46 Flowchart for Game Aggregated Review Generation*

1. Review Collection

We start by gathering a collection of existing reviews for the game. These reviews come from various sources, including players and professional critics. The goal is to have a diverse set of opinions that cover different aspects of the game.

2. Processing Critic Reviews

Among the collected reviews, we focus on those provided by critics. These reviews often offer valuable insights and analysis. Similar to the Keyword Extraction process (as discussed in Section 4.2), we process these critic reviews. The outcome is a JSON file containing a list of keywords associated with different aspects of the game.

3. Sentiment Analysis

We prompt the Mistral LLM (Language Model) to perform sentiment analysis on these aspects. This step helps us understand how positively or negatively each aspect is perceived. By analyzing the sentiment, we can identify the game's overall reception.

4. Genre-Based Topics

We prompt the Mistral LLM (Language Model) to perform sentiment analysis on these aspects. This step helps us understand how positively or negatively each aspect is perceived. By analyzing the sentiment, we can identify the game's strengths and weaknesses.

5. Generate TL;DR Summary

With information from the critic reviews, sentiment analysis, and genre-based topics, a detailed prompt was created. This prompt is then fed to the LLM, which generates a concise

summary based on the available data. The resulting summary is saved in our database and displayed on the web application for users to access easily (See Figure 47).

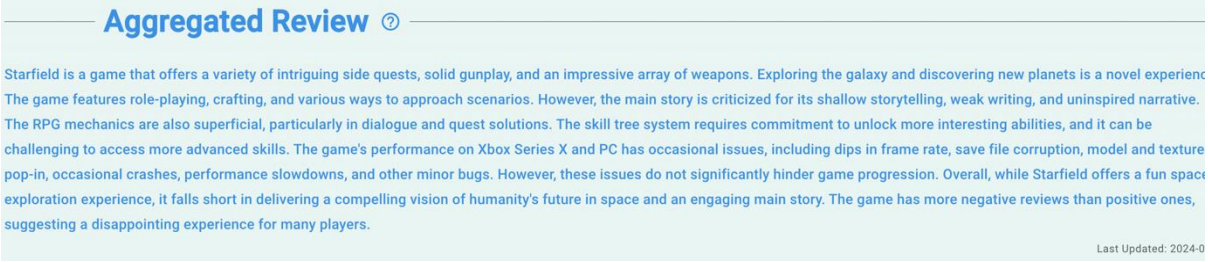



*Figure 47 Display of the Aggregated Review Summary in the Web Application's Game Page*

RAG was also used to generate the summary of a game. However, the vectorized data are stored in the persistent memory of the Docker container using Docker Volume instead of a volatile temporary vector table and will retrieve the top 5 most similar topics instead of 1 to make the summary generated more conclusive of the overall topics discussed in many of the reviews. Similar to the summary generated for individual results, generating a game aggregated review requires the analytics of the game and all of the reviews, making use of the ***gameAnalytic*** backend API endpoint which returns detailed results (See Figure 48).


*Figure 48 JSON result from gameAnalytic API*

Based on this JSON result, the sentiment analysis-related objects are extracted, including "ageReviews", "genderReviews", "sentimentReviews", "sentimentReviewsByAge", "sentimentReviewsByGender" and lastly the "topicFrequency" (the frequency of each topic based on the reviews). Afterward, the same procedure as Topic Modeling is performed with a different final LLM prompt that requests a summary using the sentiments, topics, and game aspects.

To enhance the final summary generation, the system leverages accumulated knowledge to heighten the precision and depth of information. This involves generating key insights on the posed questions, which in turn, enriches the Large Language Model's (LLM) capacity for high-level reasoning tasks. Specifically, the LLM is tasked to create 10 distinct summaries, each corresponding to a particular aspect. These summaries are then synthesized to form a comprehensive overview, rather than directly soliciting a summarization from the LLM.

Furthermore, the same trio of techniques—Topic Modeling, Keyword Extraction, and Sentiment Analysis—are employed to refine the accuracy of the summarization output. These methods work in tandem to extract pivotal themes, discern the most salient terms, and gauge the underlying sentiment of the content.

To encapsulate, the TL;DR process amalgamates essential insights, sentiment evaluation, and genre-specific data to condense an extensive game summary into a succinct narrative. This streamlined summary is prominently displayed at the top of the application's interface, offering users immediate access to a clear and accurate digest. This approach mitigates the potential frustration and tedium associated with sifting through a plethora of reviews, which may often present overlapping or conflicting viewpoints. By providing a concise and cogent summary, the system aids users in quickly grasping the collective sentiment and key takeaways from the user reviews.

### 4.4 Game Analytics

The `/gameAnalytic` API endpoint was implemented to provide comprehensive analytics for a game and the JSON object returned will be used in the analytic page in the web application to visualize the information below.

**API Endpoint: `/gameAnalytic`**

In the pursuit of enhanced performance and reduced computational demand, our API endpoint has been meticulously engineered to activate updates solely upon explicit request. This judicious design choice is underpinned by the implementation of a caching mechanism that judiciously serves a previously processed version of the game's analytics, provided that the request falls within a 24-hour window post the initial computation. This strategy is instrumental in circumventing unnecessary processing and ensuring the delivery of prompt and efficient responses. The legacy approach, predicated on Batch Processing, mandated the generation of analytics for each game at predetermined intervals. This approach, albeit systematic, precipitated a cascade of inefficiencies, notably the saturation of the database connection pool—a direct consequence of the substantial processing overhead associated with the voluminous datasets comprising reviews, game statistics, and user profiles. The ramifications were palpable: a discernible degradation in the application's responsiveness and a protracted data retrieval latency.

To address these challenges, we have transitioned to an on-demand processing paradigm. This paradigm shift is characterized by its selective processing ethos, wherein the labor-intensive task of aggregating and scrutinizing the extensive corpus of reviews associated with a game is executed no more frequently than necessitated. By adopting this approach, we ensure that the system's resources are judiciously allocated, thereby optimizing the overall user experience, and maintaining the integrity of the web server's performance. This efficient retrieval is facilitated by the **analyticUpdatedAt** field, a **DateTime** object maintained within the database table, which tracks the most recent update time. Consequently, this mechanism conserves resources by avoiding redundant data processing and serves up-to-date analytics to the end-user."

**JSON Response Breakdown**:

- number of reviews: Total number of reviews received, which can be graphed to show the game's popularity over time.

- recommendedReviews: A breakdown of recommendations, useful for a pie chart to display the proportion of positive vs. negative feedback.
- score: The average score of the game based on the reviews posted as a comparative parameter.
- Playtime: Distribution of playtime shows the length, and replayability of a game.
- percentile: The game's ranking percentile across all games in the database.
- ageReviews and genderReviews: Demographic data that can be represented in bar graphs to understand the game's audience.
- reviewLength: Insights into the length of reviews, indicating user engagement.
- sentimentReviews: Sentiment analysis of the reviews, showing the number of positive and negative reviews.
- sentimentReviewsByGender and entimentReviewsByAge: Detailed sentiment analysis categorized by age and gender.
- topicFrequency: Common topics mentioned in reviews, which can be used for a word cloud visualization.

**Graphing for Developers**:

Developers can use these analytics to:

- **Track the game's reception**: By analyzing trends in player feedback, developers can identify the most praised features as well as areas needing improvement, enabling them to refine the game based on actual user experiences.

- **Understand player demographics**: Detailed demographic data helps developers to understand who their players are, which can inform the creation of content that resonates with their core audience and attracts new players.

- **Monitor playtime metrics**: Insights into how long players engage with the game can inform decisions on pacing, difficulty, and content updates to enhance longevity and encourage replayability.

- **Evaluate sentiment trends**: Keeping a pulse on player sentiment allows developers to proactively address issues, improve player satisfaction, and maintain a positive reputation in the gaming community.

**Graphing for Users**:

Users can benefit from these analytics by:

- **Gauge game popularity**: By observing how many others are playing and enjoying the game, users can determine if it's a trending title that aligns with their interests.
- **Assess community and support**: Understanding the strength and responsiveness of the game's community and support structures can be a deciding factor in a user's purchase decision.
- **Experience insights**: Playtime and review sentiment analysis provide a window into the typical player experience, offering a preview of what new players might expect.

The **gameAnalytic** API endpoint serves as a crucial tool for delivering rich, data-driven insights. When this data is visualized (see Section 4.5.6), it transforms into actionable intelligence for both developers and users. For developers, it's a roadmap to enhancing the game's features and player engagement. For users, it's a snapshot of the game's health and community, aiding in informed decision-making. Together, these insights contribute to a more robust gaming ecosystem, where user feedback and data analytics drive continuous improvement and innovation

## 4.5 Web Application

This section explains the design, features, and implementation of the web application. The web application was developed using the technologies, framework, and design approach specified in the proposed methodology (Section 3.2). The pages and important user interface elements that have been developed in this stage are the toolbar (Section 4.5.1), register and login popup modal (Section 4.5.2), forget password and reset password pages (Section 4.5.3), landing page (Section 4.5.4), profile page (Section 4.5.5), search results page (Section 4.5.6), game page (Section 4.5.7), game analytic page (Section 4.5.8), review page (Section 4.5.9) and Progressive Web App (4.5.10) . The following sub-sections will discuss and explain the mentioned pages and components in detail. The current stage of the development has implemented the responsive web design for mobile viewport only for the toolbar and the search result page. The responsive design of other pages will be gradually implemented in the future.

### 4.5.1 Toolbar

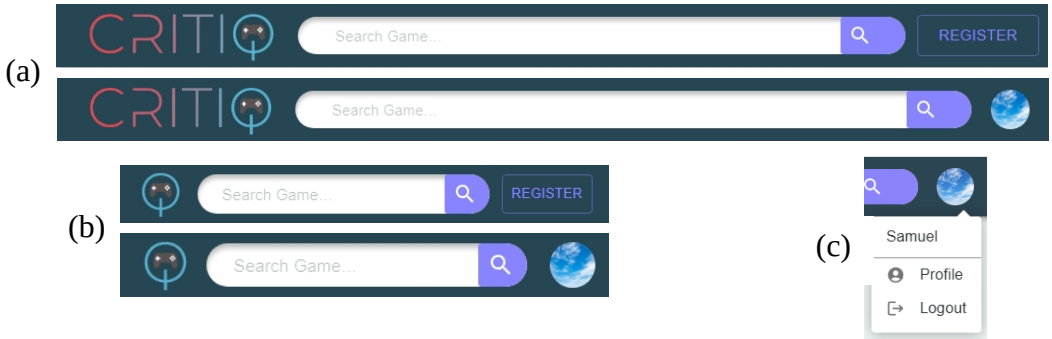



*Figure 49 Web application's toolbar design. (a): Toolbar design for desktop viewport. (b): Toolbar design for the mobile viewport. (c): Avatar icon button drop-down menu.*

The toolbar was implemented using the AppBar and Toolbar components from MUI. The layout of the toolbar was customized to suit the needs of our web application and consisted of three major sections (See Figure 49).

On the left is the app icon button that redirects the user to the landing page when clicked. In the middle, there is a search bar that allows the user to search for games with game titles. If no input is given, it will search for all the games in the database. The user can initiate the search by clicking the search button or pressing the enter key while typing the input field, after that, it will take them to the search results page.

Finally, on the right, there is either a register button or a user avatar button, depending on the user's login status. The register button opens a popup modal that enables the user

to create a new account or sign in to their existing one. The details of the popup modal's implementation, features, and design will be discussed in Section 4.2.2. The user avatar button opens a menu (see Figure 49c) that displays the username and two button options: the profile button and the logout button. The profile button takes the user to the profile page, while the logout button signs the user out of the web application.

The toolbar follows the principles of Responsive Web Design (RWD) in its design and implementation. For the viewport of mobile devices (See Figure 49(b)), the app icon button is substituted by a simplified version of the icon, the spacing between the three components is minimized, and the dimensions and font size of the register button are scaled down. These measures ensure that the toolbar can adapt to the smaller viewport and offer the optimal user experience.

The toolbar also adopts a dynamic display strategy to enhance the website's visual clarity and information density. The toolbar disappears when the user scrolls down and reappears when the user scrolls up.

### 4.5.2 Login and Registration

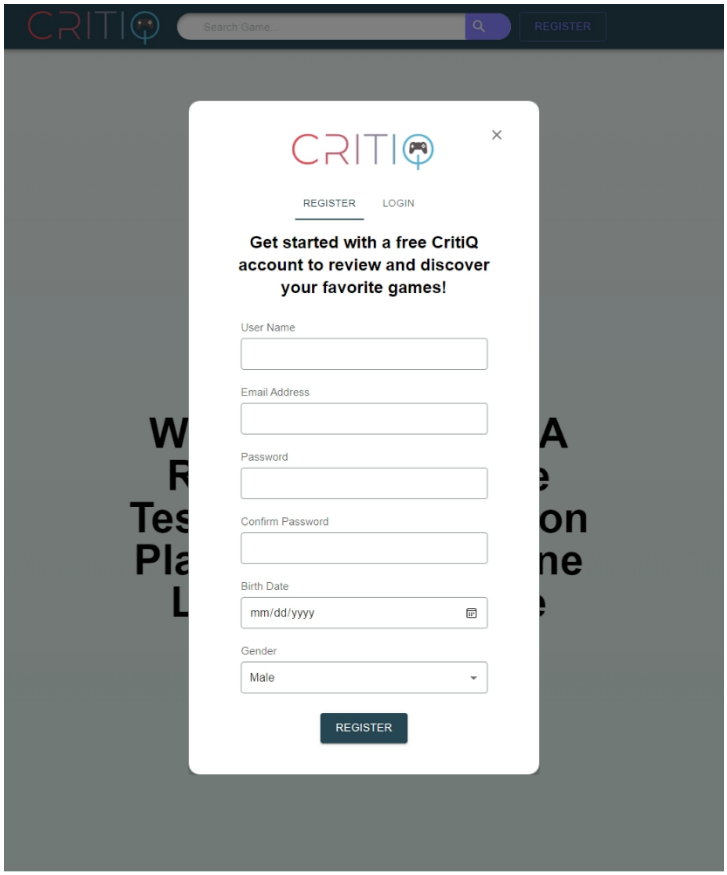



*Figure 50 Web application's register modal popup*

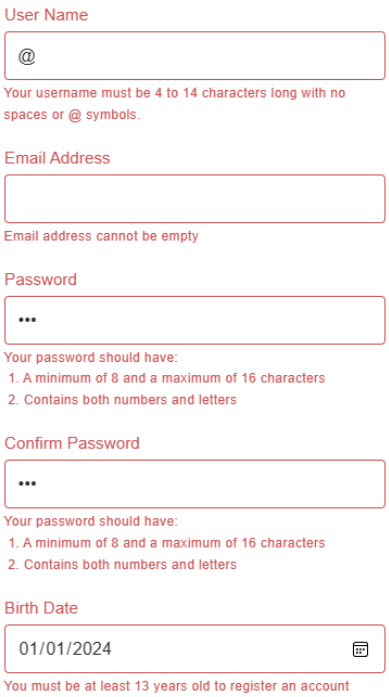



*Figure 51 Register modal input validations*

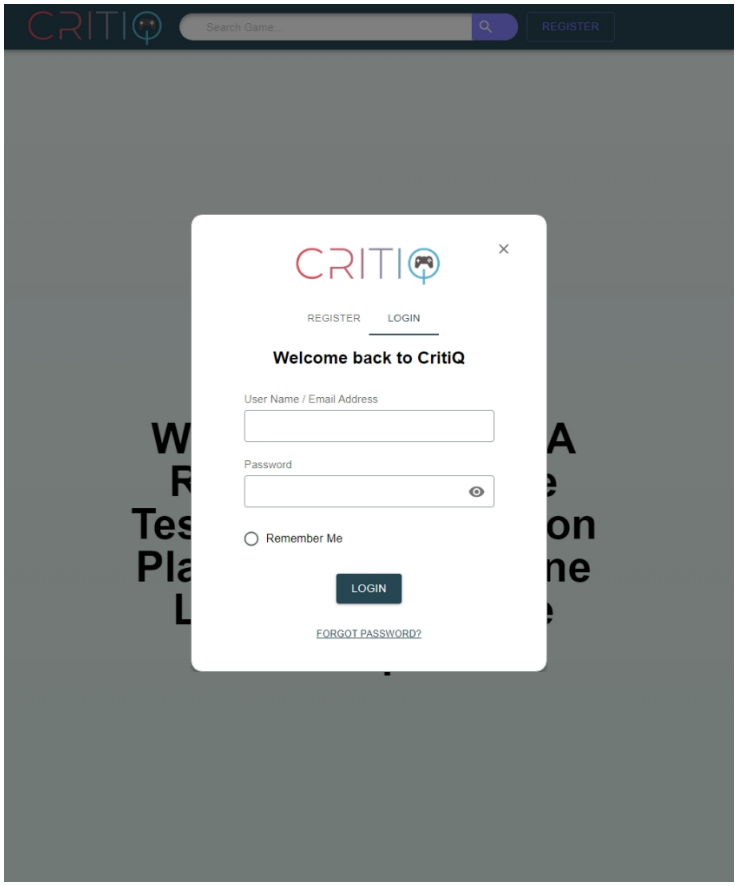



*Figure 52 Web application's login modal popup*

As mentioned in the previous sections, the popup modal can be accessed by the users by clicking the register button on the toolbar. This modal enables the users to either register a new account or log in to their existing account. The Modal component from

MUI is utilized to implement this popup modal and the input fields are a customized version of the InputBase component from MUI. The layout of the popup modal consists of the web application icon and an icon button to close the modal on the top, and the tab bar below the icon. The Tabs and Tab components from MUI are employed to implement the tab bar. The users can toggle between the register modal and the login modal by selecting the corresponding tab.

The register modal (See Figure 50) requires the users to enter all the fields to create a new account, which comprises username, email address, password, confirmed password, birth date, and gender. The front-end application performs validations on all the input fields using regular expressions (See Figure 51) based on the following 2 rules. This is crucial as it reduces the network load for the backend system by preventing the users from making invalid requests using the frontend application.

1. The username must have a length of 4 to 14 characters with no spaces or @ symbol, as this symbol is designated for email address detection.
2. The password must have a length of 8 to 16 characters with both numbers and letters to ensure its security.

The backend system also validates the register request to avoid duplicate usernames or email addresses in the database.

The login modal (See Figure 52) requires the users to enter the username or email address and password to sign into their account. The icon button on the right of the password input field allows the users to hide or show the password by changing the type of input field between text and password. The “Remember Me” checkbox below the two input fields determines how the refresh token cookies are stored, as explained in Section 3.2.2. The backend system is responsible for the validations, and it will return an error message with a description of the issue for the invalid login request. The modal will display this error message to inform the user. If the user has forgotten their password, they can click the forget password button below the login button, which will redirect them to the forgot password page. The next section will describe the implementation of this page.

### 4.5.3 Forgot Password Page and Reset Password Page

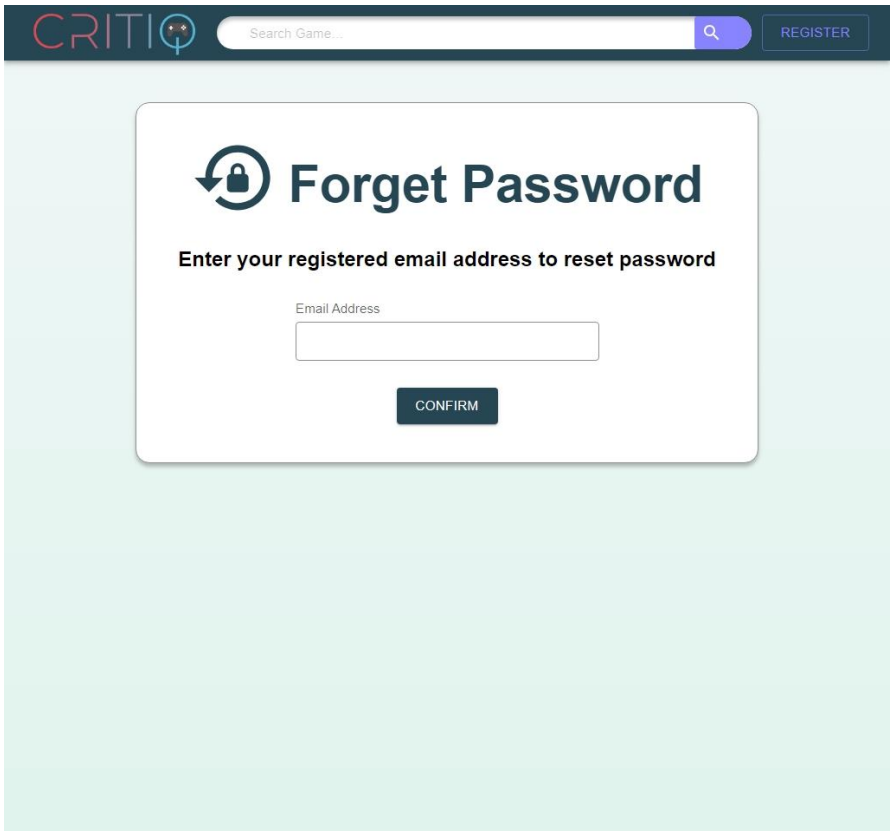



*Figure 53 Web application's forget password page design.*

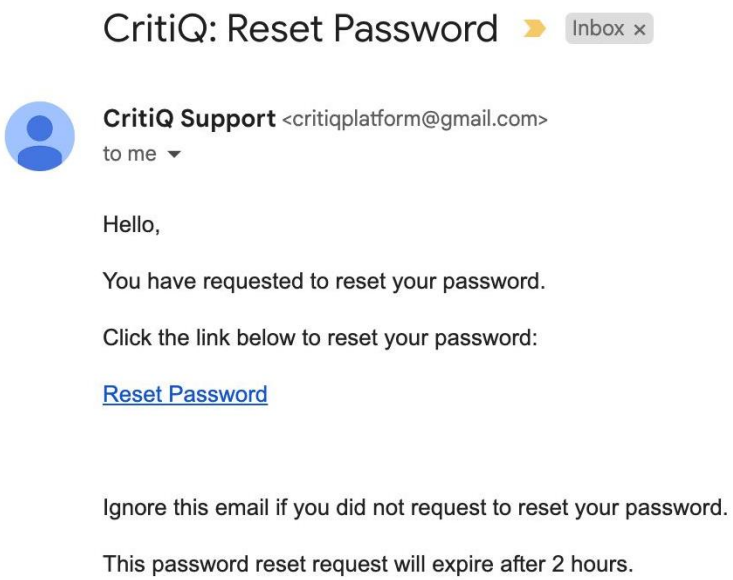



*Figure 54 Reset password email*

The user can access the forgot password page (See Figure 53) by clicking the forgot password button on the login page. On this page, the user can input the email address associated with their account. Validations will be performed by both the frontend application and the backend system. The frontend application uses regular expressions

to validate the email address format, and the backend system queries the database to check the email address's existence. If the email address is valid, the system generates an email with a link to the reset password page (See Figure 54) and sends it to the user after they click the confirm button. The link contains a unique token that corresponds to a specific account, and this token is passed as a prop to the reset password page to enforce authentication and authorization. The backend system also persists the token in the database for verification purposes and sets it to expire after 2 hours for increased security. To prevent email spamming, the web application disables the confirm button for 60 seconds after sending the email.

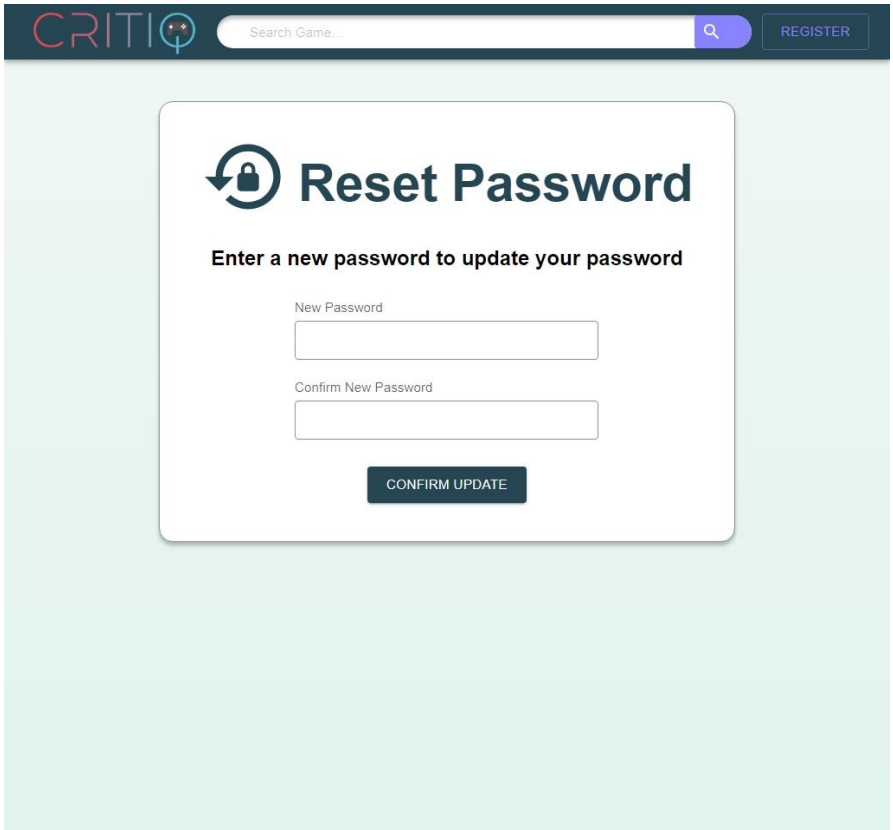



*Figure 55 Web application's reset password page design*

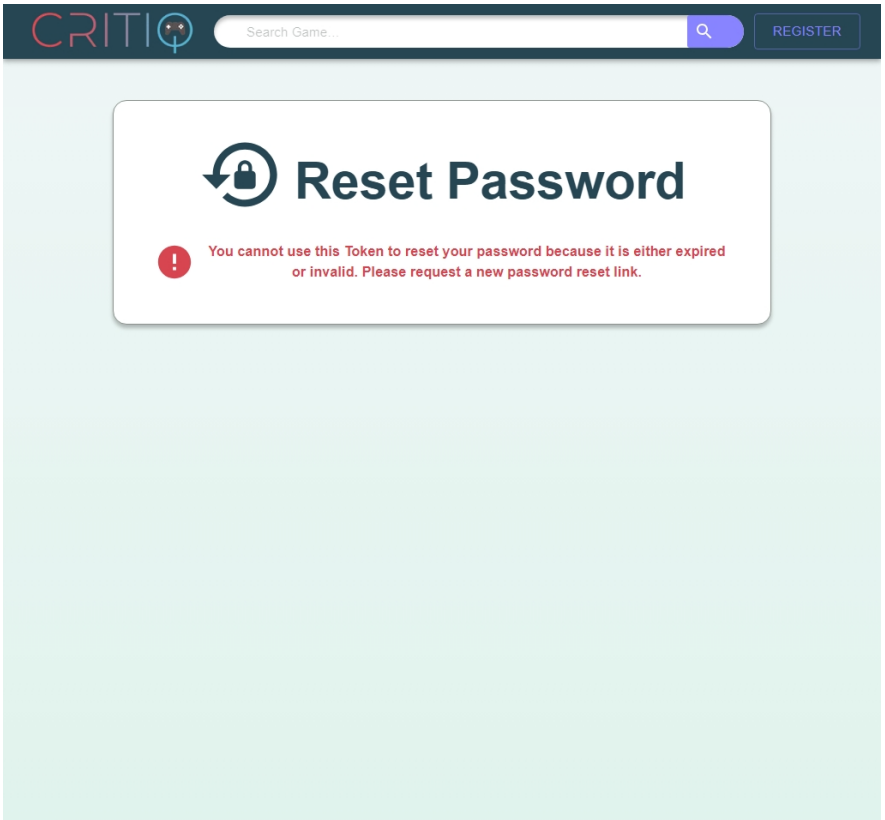



*Figure 56 Web application's reset password page with invalid token*

On the reset password page, the web page displays a form to reset a new password (See Figure 55) if the token in the link is valid. Otherwise, an error message is shown to inform the user of the invalid or expired token (See Figure 56). To update the password, the user must enter a new password and confirm it in the respective input fields in the form. The system enforces the same password rules as for registering a new account. After the user clicks the confirm update button, the web application sends the update password request to the backend system. Validation is performed by the backend system to ensure that the new password is different from the old password. If the request is invalid, the backend system will return the error to be displayed by the frontend system, otherwise, the backend will modify the password in the database.

### 4.5.4 Landing Page

The landing page is the first page the user will visit when going to the web application and can be accessed by clicking on the Logo in the App bar at any time(See Figure 56).

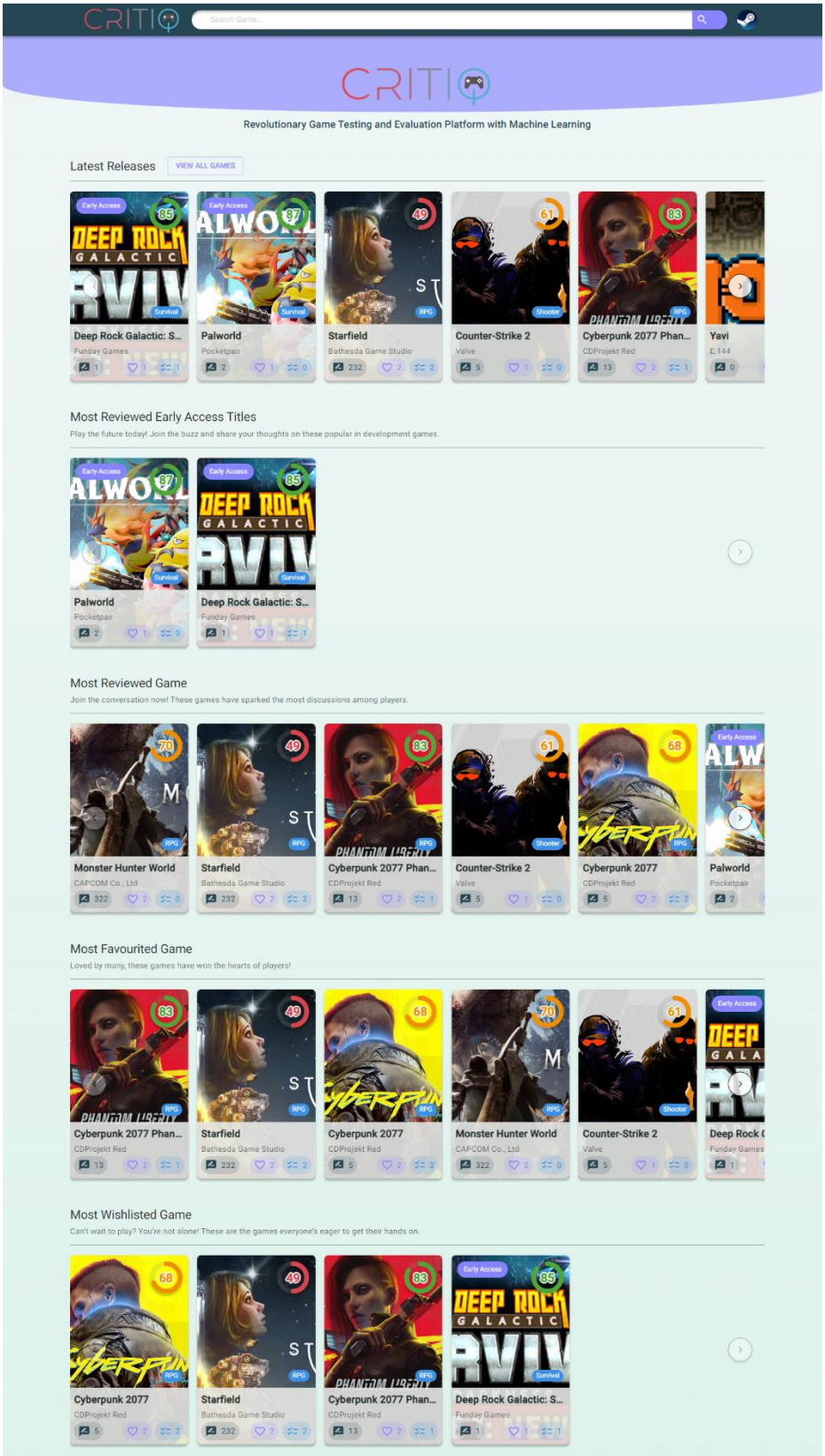



*Figure 57 Landing Page of the Web Application, showing the logo, short descriptions along with 5 carousels showing popular titles*

The landing page contained 2 primary sections:

The header section displays the application's name alongside a summary of its purpose. While the "Hottest" section where there will be 5 carousels, one for each of the following:

1. Top Most Reviewed Games Carousel: Dive into the games that have sparked the most conversation in the gaming community. This carousel features titles that have received the highest number of detailed reviews, reflecting their impact and popularity.
2. Top Favourited Games Carousel: Discover the games that players love the most. This section highlights the games that have been favourited by users, indicating a strong approval and recommendation from our gaming community.
3. Top Wish listed Games: Explore the games that are on everyone's Wishlist. These are the titles that users are eagerly waiting to get their hands on, signalling high anticipation and interest.
4. Latest Releases: Stay up-to-date with the newest games hitting the market. This carousel presents the freshest releases, so you're always in the know about the latest and greatest in gaming.
5. Most Reviewed Early Access Games: Get a sneak peek at the future of gaming. This section showcases early access games that have generated significant buzz and constructive feedback from early adopters.

Each of these carousel's games are fetched with their own individual API, fetching the latest results based on their own criteria, showing the real-time change of these information, including favorites or wish lists count or number of reviews.

In each of these carousels, games would be represented by game cards (See Figure 58), showing important information about the game, including the name, genre, score, number of reviews and number of wish lists and favorites. User can learn more about the game by clicking on the game card which will redirect the user to the respective game page. In particular, games in early access or in development will have a purple text box displaying the text "Early Access" at the top left of the card (See Figure 59).

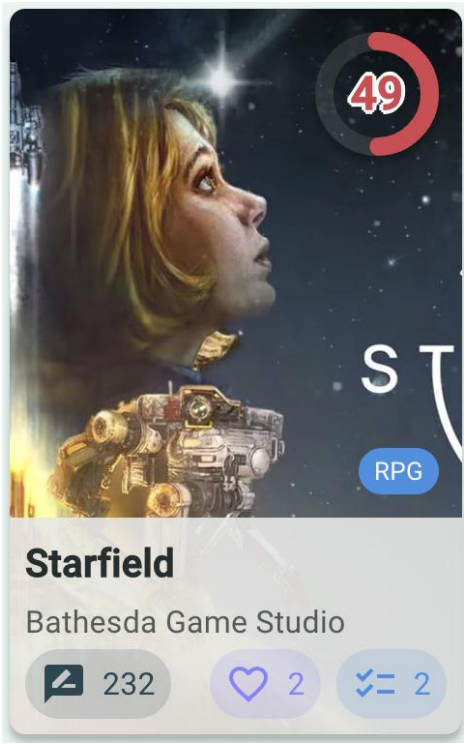



*Figure 58 Game Card within Landing Page carousel, showing the game "Starfield", along with its score, number of reviews, favorites and wish lists*


*Figure 59 Game Card showing the Early Access game "Palworld", with a purple box displaying "Early Access"*

If the current viewport cannot display all the game cards, the arrow navigation buttons will be enabled, signaled by their increased opacity and can be clicked to scroll the carousel horizontally (See Figure 60). To prevent unintended clicking, dragging of the

carousel has been disabled on desktop viewport to reduce accidental navigation to the game pages since they are implemented as clickable buttons.

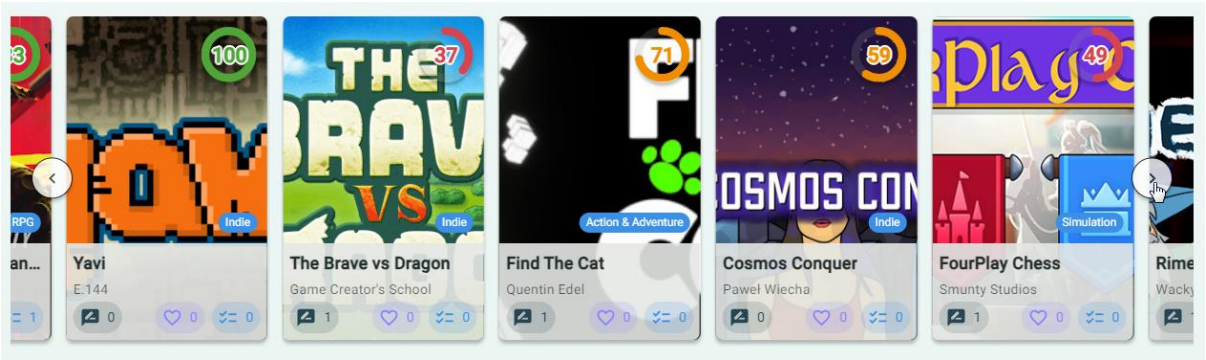



*Figure 60 Carousel with games card and arrow control to navigate the carousel horizontally*

In a mobile viewport, the navigation buttons will be disabled and user can navigate the carousel; through dragging on the game cards directly to reduce the chance of misclicking of the game card when using buttons (See Figure 61).

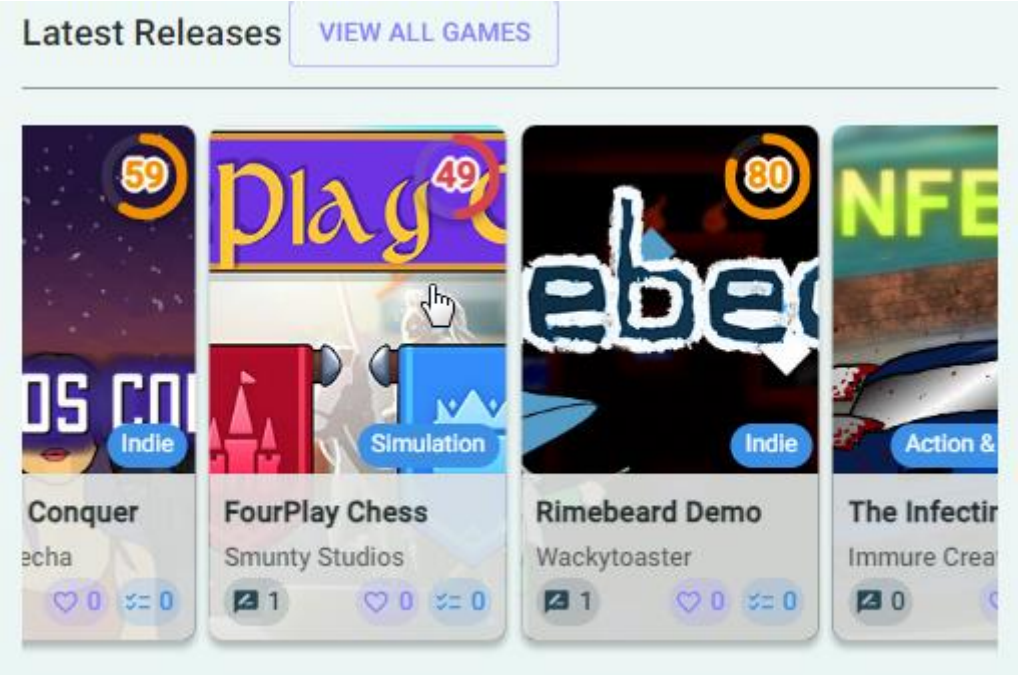



*Figure 61 Landing Page in a Mobile Viewport with navigation buttons disabled and controls replaced by dragging gestures*

### 4.5.5 Profile Page

User can visit the profile page of their own account by clicking on their avatar in the Toolbar or the name of any account in review pages. If a user is the profile owner, the privacy toggle will be visible, allowing the user to toggle the privacy of their account information including their email, gender, age, last-active date and their reviews (See Figure 61Figure 62).

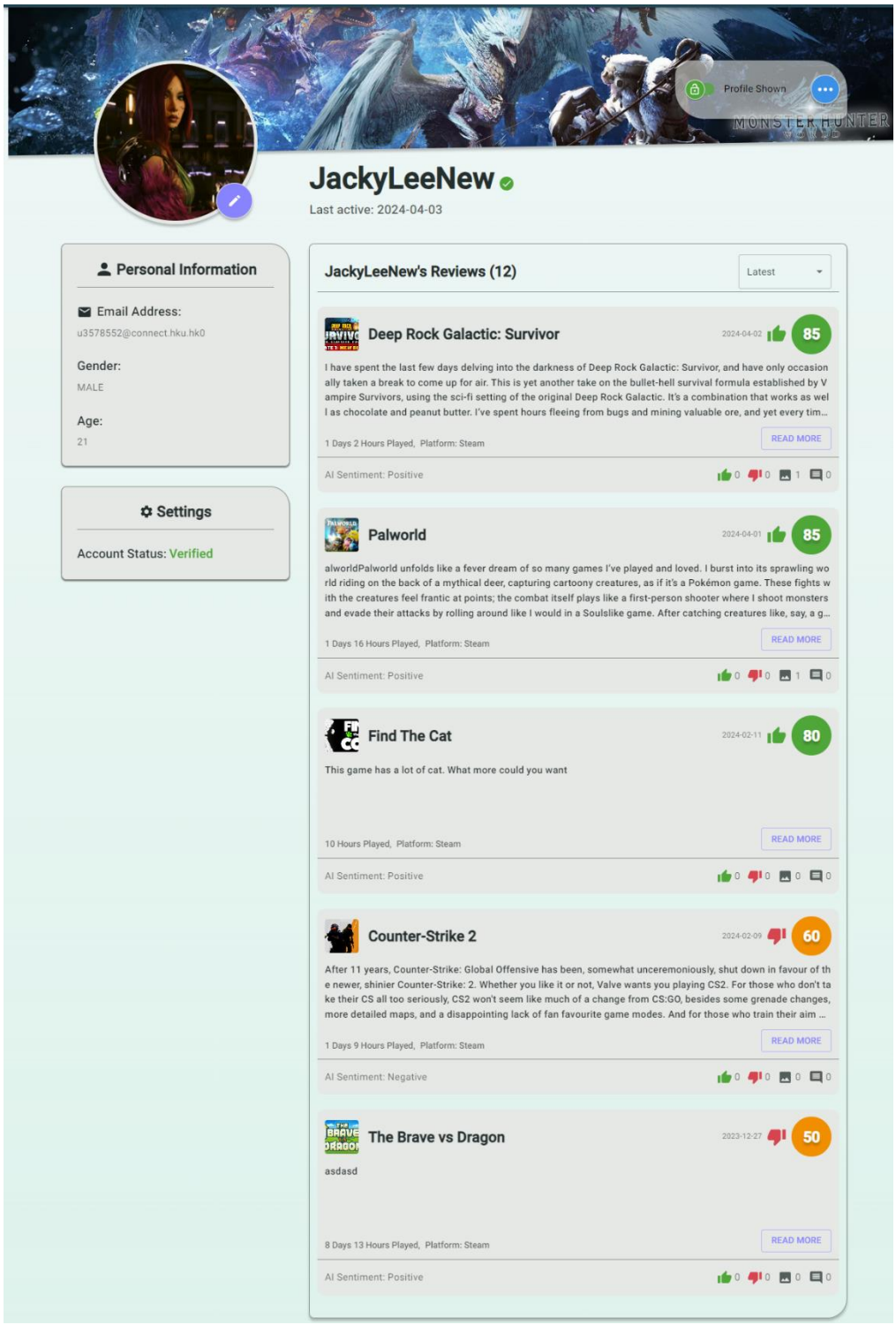



*Figure 62 Profile Page of a Logged In User, showing the ability to toggle the privacy of their account at the top right*

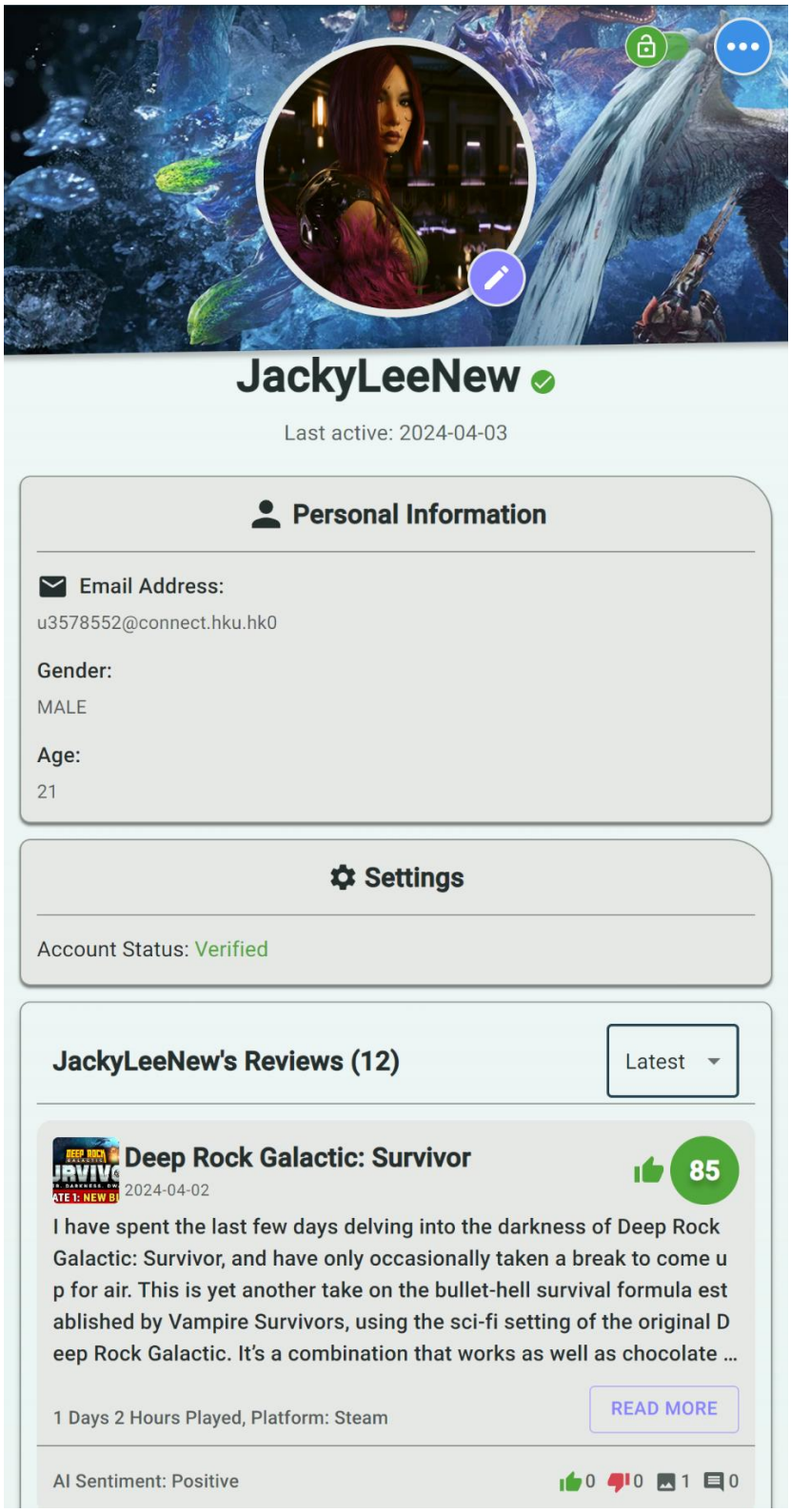



*Figure 63 Profile Page of a Logged In User in a mobile viewport, where elements are stacked vertically*

If a user visits a private account, all of the previously mentioned information will be hidden and displayed as "Undisclosed" to protect user's privacy (See Figure 64).

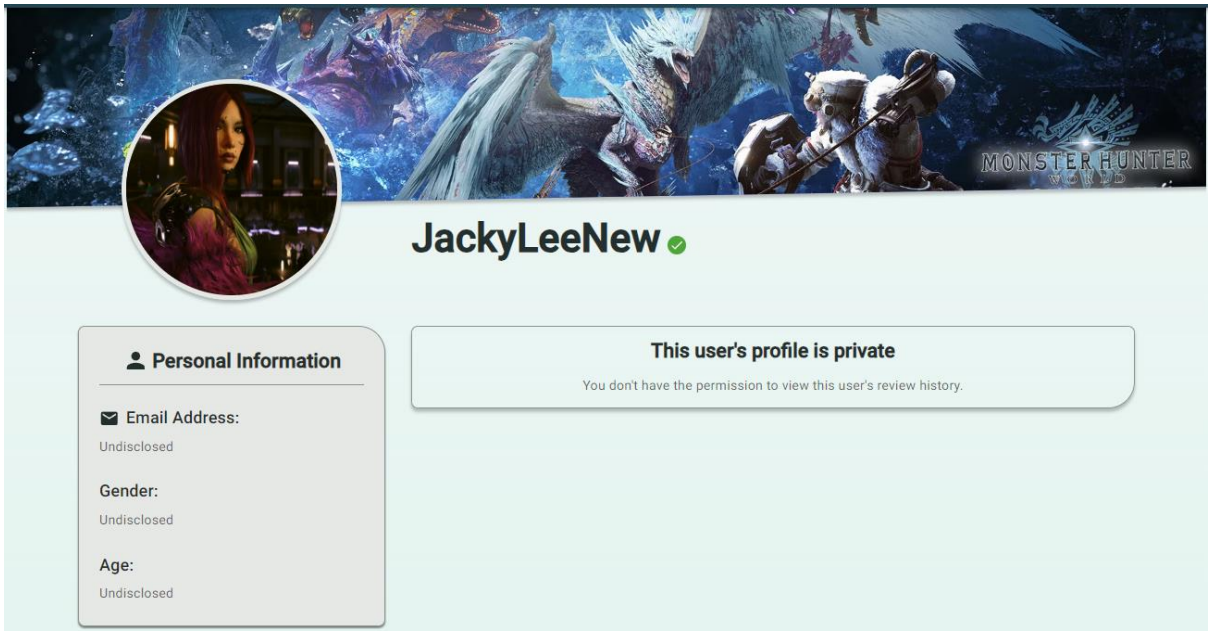



*Figure 64 Profile page of a Private User account with all information except name hidden*

Account owners can also update their username and their profile banner by clicking on the 3-dots button next to the privacy toggle (See Figure 65) and allow for quick preview before submission of the username and profile banner (See Figure 66).

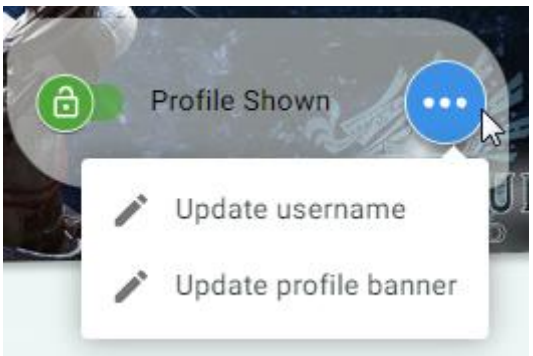



*Figure 65 3-Dots Button revealing the username and profile banner update button to the account owner*

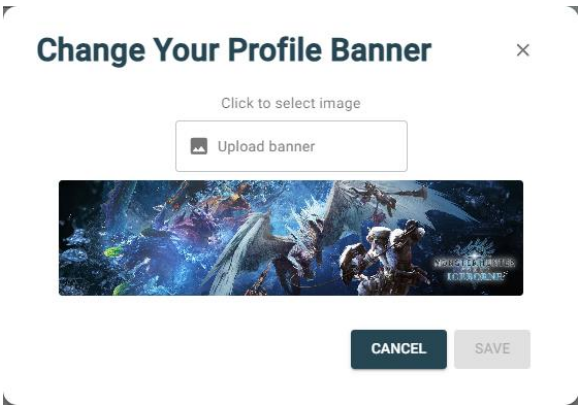



*Figure 66 Profile Banner update UI, allowing user to preview prior to confirmation*

Verified account will have a green checkmark next to their name to signify that the account owner have verified their email address (See Figure 67).


*Figure 67 Verified Account with a Green Checkmark that displays "Verified User" on hover*

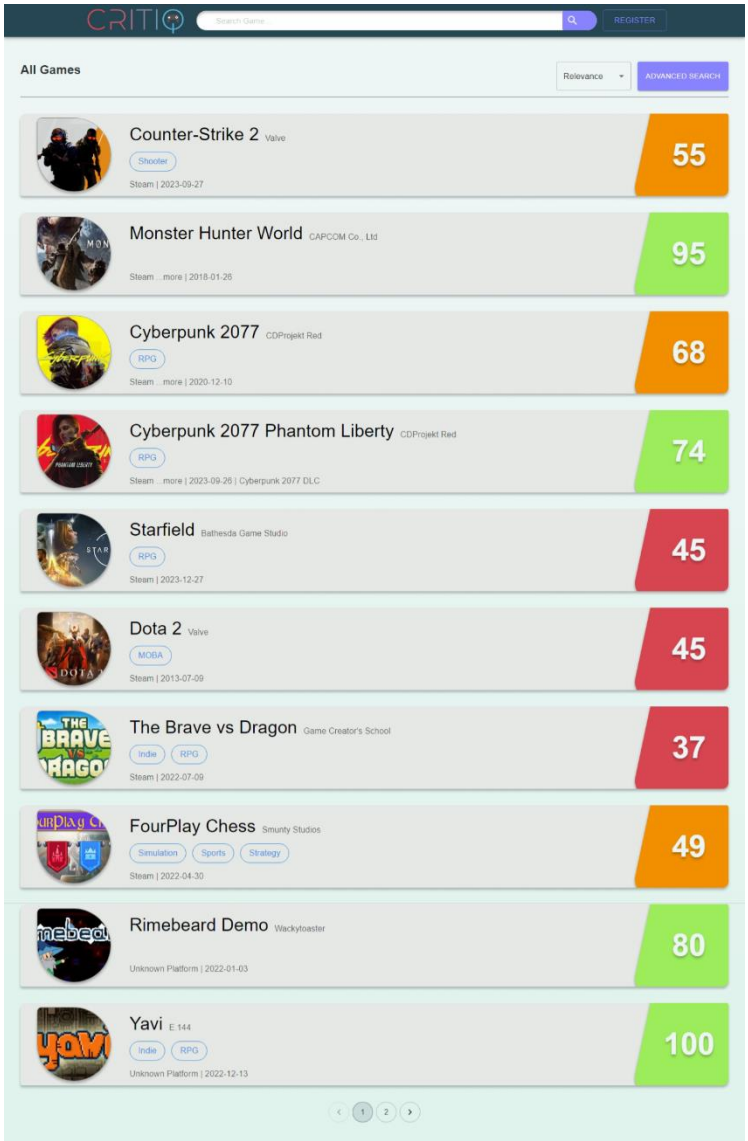
If a user's profile is public, all of their reviews will be publicly visible, along with the total number of reviews created. By default, they are sorted in terms of recency and the sorting order can be changed by clicking on the button in the review section. Two other sorting methods, namely Oldest and Highest Score can be easily accessed within the button (See Figure 68).


*Figure 68 Default sorting order of "Latest" can be changed by clicking the button, revealing two other options*

When the page is initially loaded or the sorting method is updated, at most 5 reviews will be shown, user can access the remaining reviews by scrolling down to the end of the list which will trigger the API to fetch more reviews and update the list in the UI until no more reviews are available and a text showing "*NO MORE REVIEWS TO SHOW*" will be displayed.

### 4.5.6 Search Result Page

(a)




(b)

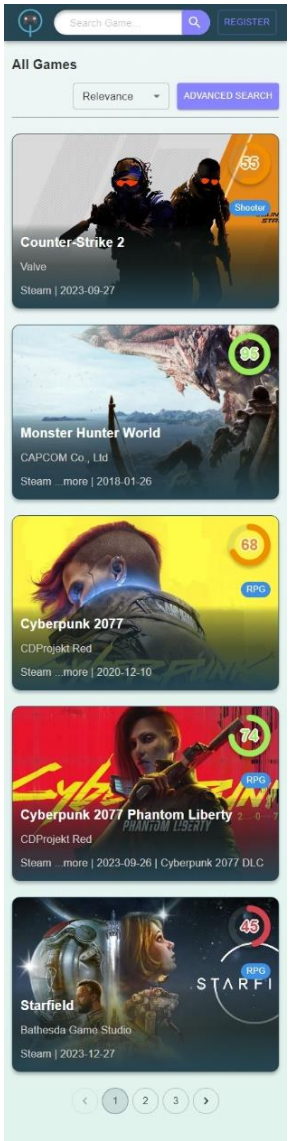



*Figure 69 Web application's search result page design. (a): Search result page design for desktop viewport. (b): Search result page design for mobile viewport.*

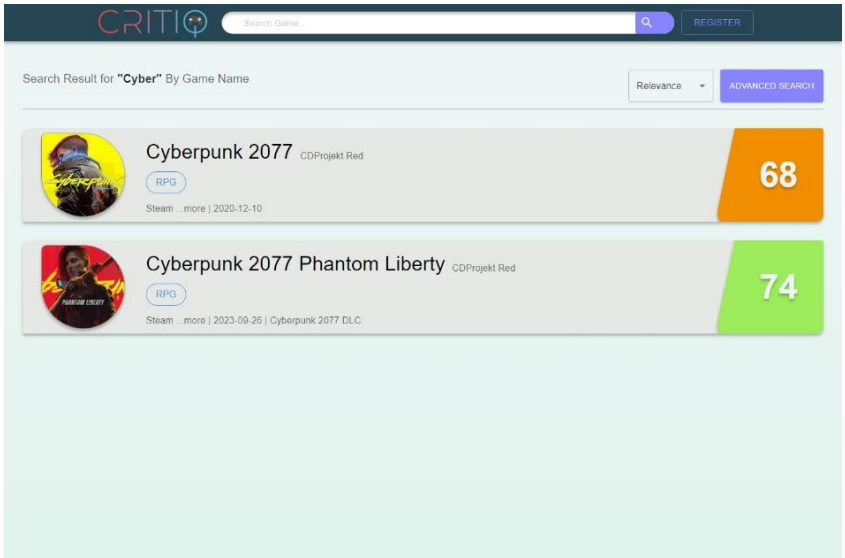



*Figure 70 Search result page searching example with game name consisting "Cyber" returning 2 results*

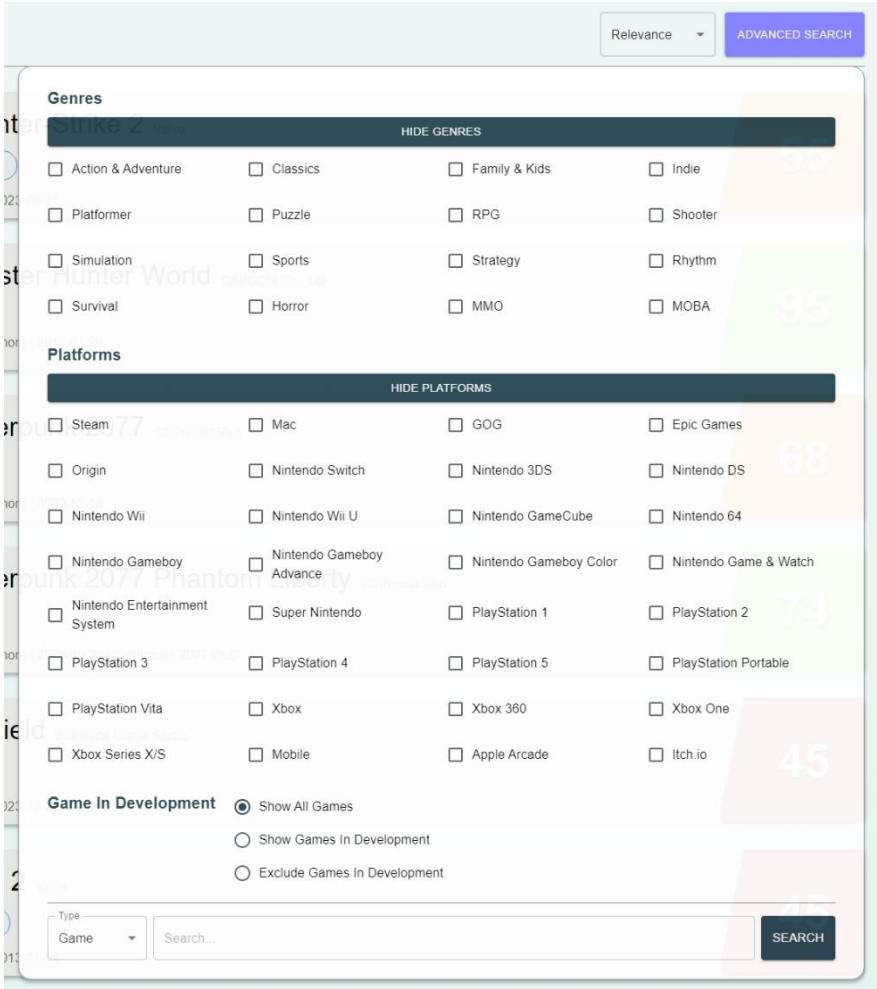



*Figure 71 Advanced search modal for search result page*

Section 4.2.1 describes how the user can access the search result page via the search bar and the search button in the toolbar. The search button has a conditional functionality based on the state of the search bar. If the search bar is empty, the search button redirects the user to the search result page that displays all games in the database (See Figure 68 (a)). If the search bar has an input, the search button redirects the user to the search result page that displays the games that match the input (See Figure 70). The default search method for the search bar is search by game name.

The search result page has a description at the top that specifies the search method and the search input used to generate the results. Three different search methods are available to the user through the advanced search feature: all games search, search by title, and search by developer. The description changes accordingly to reflect the chosen search method and input.

The user can see a select button and an advanced search button on the right of the description. The select button opens a menu that allows the user to choose the sorting method available for the game results: relevance, score, or release date. The advanced search button opens a popup menu (See Figure 71) that enables the user to perform an advanced search by applying various filtering criteria, such as genres, platforms, and development state. The user can also choose between two search methods in this menu, which are search by title or developer.

The front-end application determines the search type and filtering criteria based on the query parameters appended to the URL and sends a search API request with the appropriate body. The query parameter **gamename** is used for search by title, while **developername** is used for search by the developer. Other query parameters are **genre**, **platform**, and **isInDevelopment**, which are incorporated in the body of the API request to retrieve the filtered game results.

A URL example that searches by the developer's name "valve", with the genre of shooter, the platform of Steam, and the exclusion of games in development is https://critiq.itzjacky.info/result?developername=valve&genre=7&platform=0&isInDevelopment=false. In this example, the genre of shooter is mapped as 7 and the platform of Steam is mapped as 0 by the application. By storing variable states in the URL, these URLs can be easily copied and shared to reproduce the same search result without needing manual input again.

The game search results are displayed below the description. Each search result card component displays the basic information of the game if it exists in the database. This information includes the game icon, game name, developer name, game genres, game platforms, development state, and game release date. Additionally, the score of the game is also displayed, which is computed by the average score of all reviews. The game is classified as bad, average, or good based on its percentile rank among all games in the database. Games ranked above the 75th percentile are considered good, games ranked below the 30th percentile are considered bad, and games ranked between the 30th and 75th percentile are considered average. Good games are displayed with a green score, average games are displayed with an orange score, and bad games are displayed with a red score. Clicking on the search result card will redirect the user to the game

page, the implementation and design of the game page will be explained in the next section.

The pagination at the bottom of the page allows the user to navigate through the search results. This is implemented using the Pagination component from MUI. The number of results shown on each page depends on the viewport layout. For the desktop layout, up to 10 results are shown on each page, while for the mobile layout, up to 5 results are shown on each page.

The search result page adapts to the user's viewport by displaying the layout of the page and the search result card component differently to accommodate the various screen sizes. The font size for the mobile viewport is also reduced to ensure that all information is displayed properly (See Figure 68 (a) & (b)). This design approach ensures that users will have the optimal user experience regardless of the devices used.

### 4.5.7 Game Page

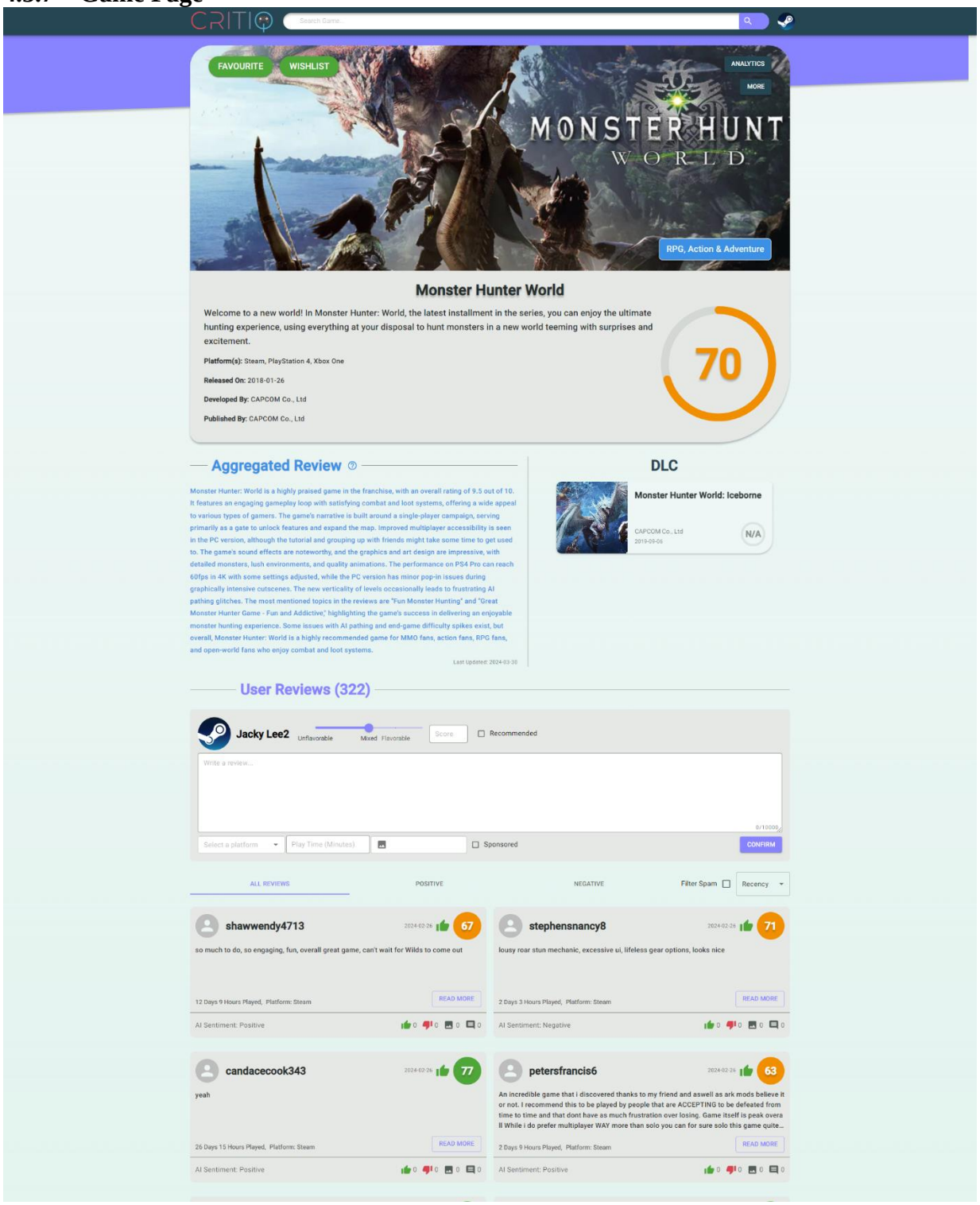



*Figure 72 Web application's game page design*

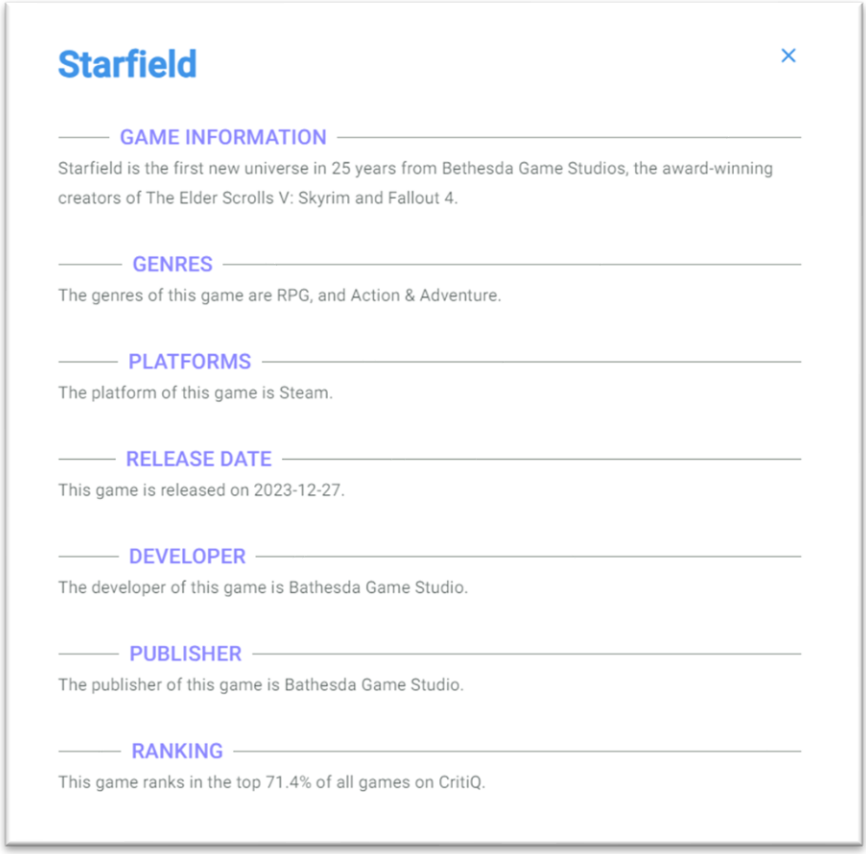



*Figure 73 Game detailed information popup modal*

As stated in the preceding section, the game page (See Figure 72) can be accessed by selecting the search result card on the search result page and includes three main sections, Information, Add Review, and All reviews.

The information section, located at the top of the game page, displays all the game information that was previously presented on the search result card. Moreover, it provides a brief overview of the game, the name of the publisher, and a comprehensive list of platforms that support the game. The brief overview is restricted to three lines, and the overflown text will be concealed by ellipsis. To access the full description of the game, the user can click on the more button located at the top right of the information section, which will open the game's detailed information modal (See Figure 73).

Three buttons, namely “Favorite” , “Wishlist” and “Analytics” had been implemented, where authenticated user can favorite and wish list games that they are interested and liked. If an authenticated user interacts with the favorite or wishlist button, a Snackbar component will be displayed displaying “You must be logged in to wishlist/favorite a

game" (See Figure 75). These favorites and wish lists will be associated with the user and be used for analytic purposes (See Section 4.5.8). These buttons will turn red if user has already interacted with these button, clicking on them again will unfavorite and unwishlist these games, turning them back to green (See Figure 74). The Analytics button is located above the More Info button which when clicked will direct to user to the analytics page for this game (See Section 4.5.8).


*Figure 74 Favorite and Wishlist Buttons will turn red if user has already favorited and wish listed this game*

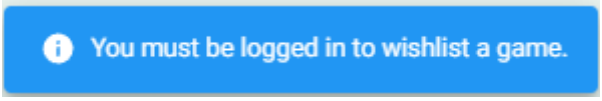



*Figure 75 Snack bar display for unauthenticated user trying to favorite or wish list a game*

The information section is followed by the aggregated review section and the DLC section. The aggregated review section will only be available to games with a sufficiently large review dataset (See Section 4.3), utilizing natural language processing models to generate an aggregated review that summarizes all the reviews of the game and data collected for analysis for the user and display the summary information in this section along with the date of the summary being generated. The DLC section will only be visible for games that have DLC (See Figure 72), and all the DLCs will be presented in a slider format with an individual DLC card. The DLC card shows the name, developer, release date, and score of the LDC, clicking on the card will redirect the user to the game page of the DLC. The slider is implemented using the React Slick library, a popular React carousel that offers a simple, lightweight, and customizable carousel component.

Following the aggregated review section and the DLC section is the user review section. This section consists of two sub-sections, which are the add review section and the game review section.

The add review section is only visible to authenticated users. Users can create new reviews by completing the input fields in the add review form (See Figure 72). They are required to provide mandatory information, including a numerical score (from 0 to 100), the recommendation status, the source of the game (self-bought or sponsored), the review content, the platform, and the total playtime. Additionally, users can optionally attach images to their reviews through the file input field, this is implemented using the MUI file input component from the MUI file input library. The submitted images are limited by quantities and size, with a maximum of 10 images and a size of less than 3 Megabytes per image. This constraint aims to prevent users from uploading an excessive number of high-quality images to our storage bucket and hindering the performance of page load. When the user clicks on the confirm button, both the frontend and backend perform validations to ensure that all required fields are filled, and the attached images do not surpass the size limit. If the new review is successfully created, the web application will redirect the user to the review page.

If user has already submitted a review for that game, the add review section will be replaced by the user's existing review, allowing user to quickly view their review (See Figure 76). If there was no previous edit on the review or the last edit date was more than 1 week ago, an "Edit Review" button will be available to the user, allowing the modification of their review, score, recommendation, playtime while disallowing the edit of the reviewed Platform and images.

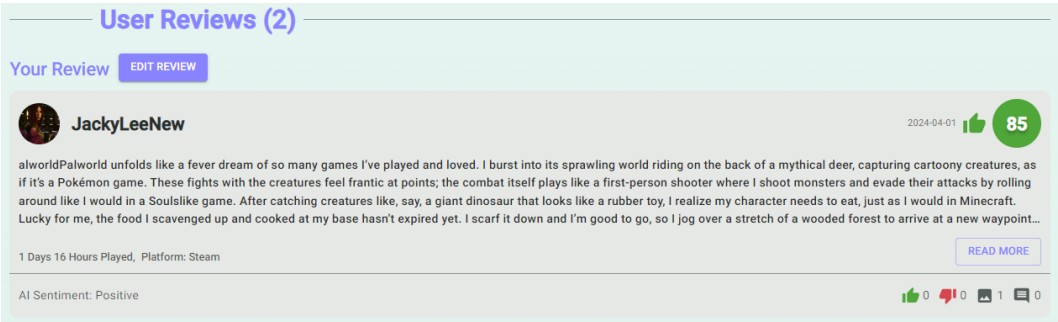



*Figure 76 User with existing review will not be able to create new review and can view their current review and have the ability to edit their review*

Clicking on the “Edit Review” button will reveal the edit review section, where user can edit the information mentioned above and submit their updated review and will be redirected to the review page on submission, showing an “Edited At” time in the review section (See Figure 77, Figure 78). Upon submission of the edited review, all review-related NLP tasks will be retriggered to reflect on the edited review. A time until next edit will be displayed, replacing the Edit Review button, for reviews that were edited within one week of the time of viewing, reducing the operation cost of rerunning all NLP-related processes.

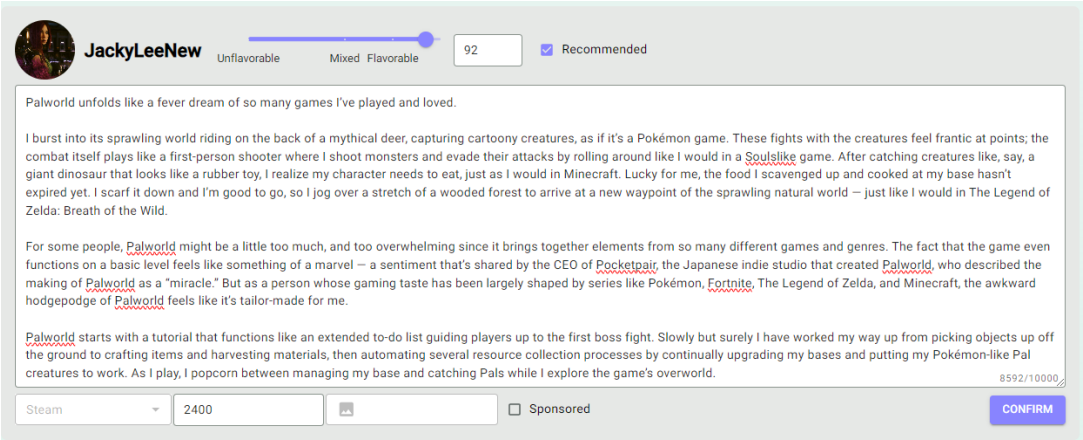



*Figure 77 Review Section change to edit review when user click on the "Edit Review Button"*


*Figure 78 Review page with "Edited At" time showing the time of review being edited*

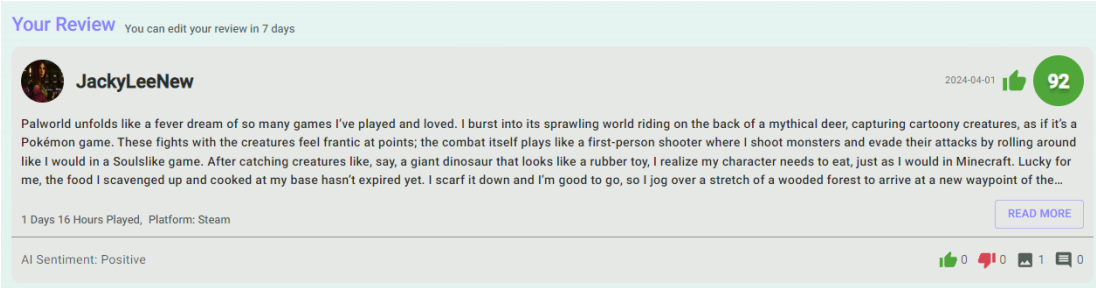



*Figure 79 Edit Review button hidden with text displaying Time until next available edit.*

The game review section exhibits the reviews that other users have written for the game. A tab bar for filtering and a select button for sorting are located at the top. Users can choose to display all, negative-only, or only positive-only reviews by selecting the respective tab in the tab bar. Clicking the select button opens a menu for users to select the sorting criterion for the reviews, which comprises recency or score. The game

review cards, which are components that present some essential information about the game review and the reviewers, are situated below the tab bar and the select button. The first row in the game review card displays the avatar icon and name of the reviewers, the date and time the game review was created, a thumbs-up or thumbs-down icon indicating the recommendation status of the review, and the score that the reviewer assigned to the game. The score color is computed dynamically: scores above 75 are deemed as good and shown in green, scores below 50 are deemed as bad and shown in red, and scores between 50 and 75 are deemed as average and shown in orange. The second row shows the review content, which is limited to four lines, and the excess content is concealed by ellipses. The left side of the third row reveals the total playtime, the platform, and the game version that the reviewer played the game on, and the right side has the read more button, which will redirect the user to the review page upon clicking. The review page will be explained in detail in the subsequent section. Lastly, the fourth row displays the sentiment result of our NLP model on the left, and the number of likes, dislikes, images, and review comments on the right. The game review cards are structured using the Grid component from MUI, which enables the adjustment of the card per row according to the viewports. The desktop layout displays two game review cards per row, while the mobile layout shows one game review card per row.

### 4.5.8 Game Analytic Page

The Game Analytic Page can be accessed by pressing the Analytics Button on the Game page. The data shown on the page will be updated whenever a user requests for the data and the data are more than 1 day old, which helped reduce the performance overhead of constantly fetching all game-related data and doing the data analytics.

The graphs displayed on the page make use of the *Nivo* Graphing Library which is built on top of D3.js (See Figure 80, Figure 81), providing responsiveness, higher performance, and reducing rendering latency compared to alternatives such as Chart.js that make use of the canvas, leading to performance issues, particularly on slower devices or browsers using the Firefox's Quantum Engine due to its slow canvas updating.

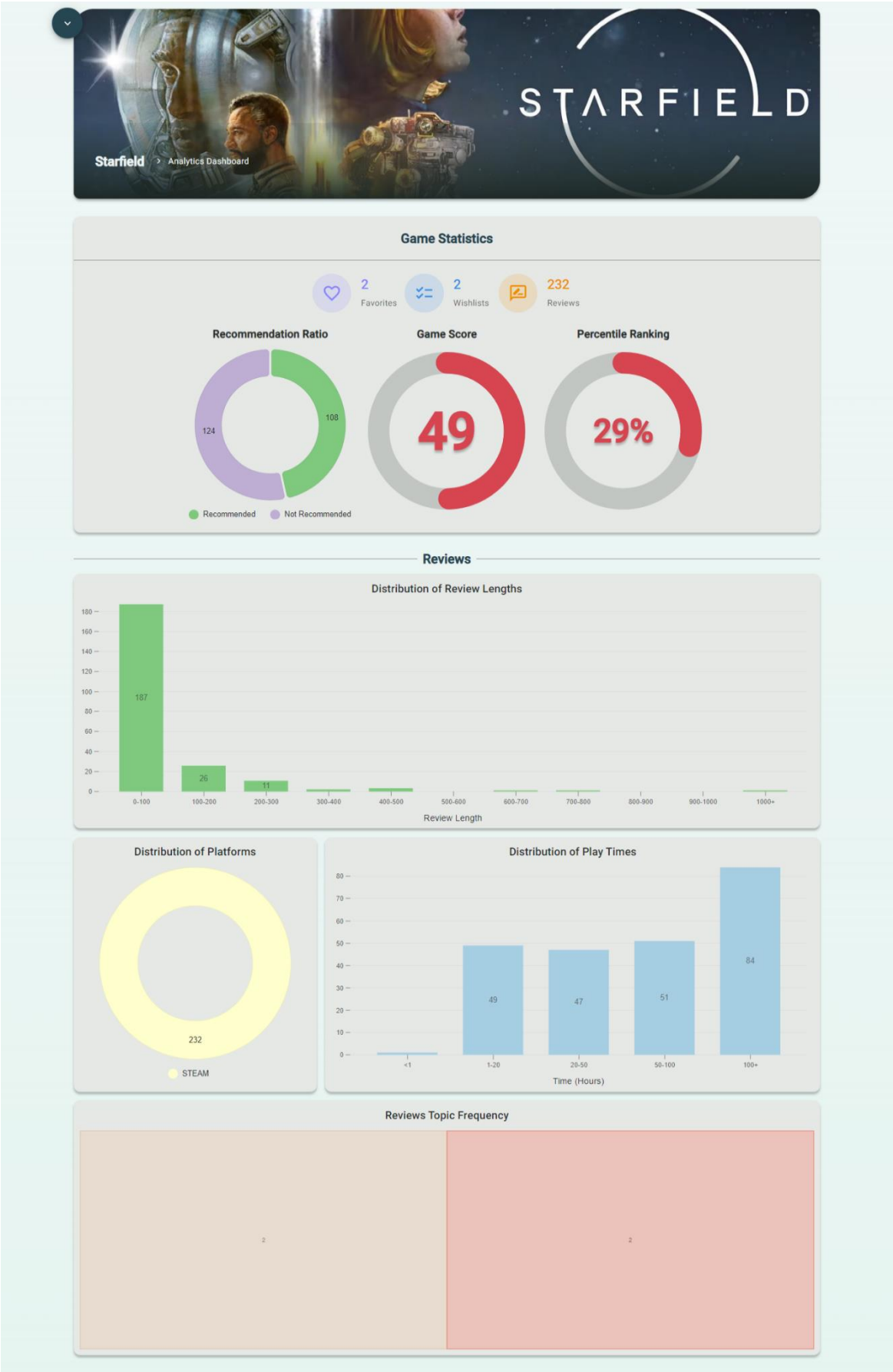



*Figure 80 Game Statistics and Review Statistics of the Game Starfield in the Analytics Page*

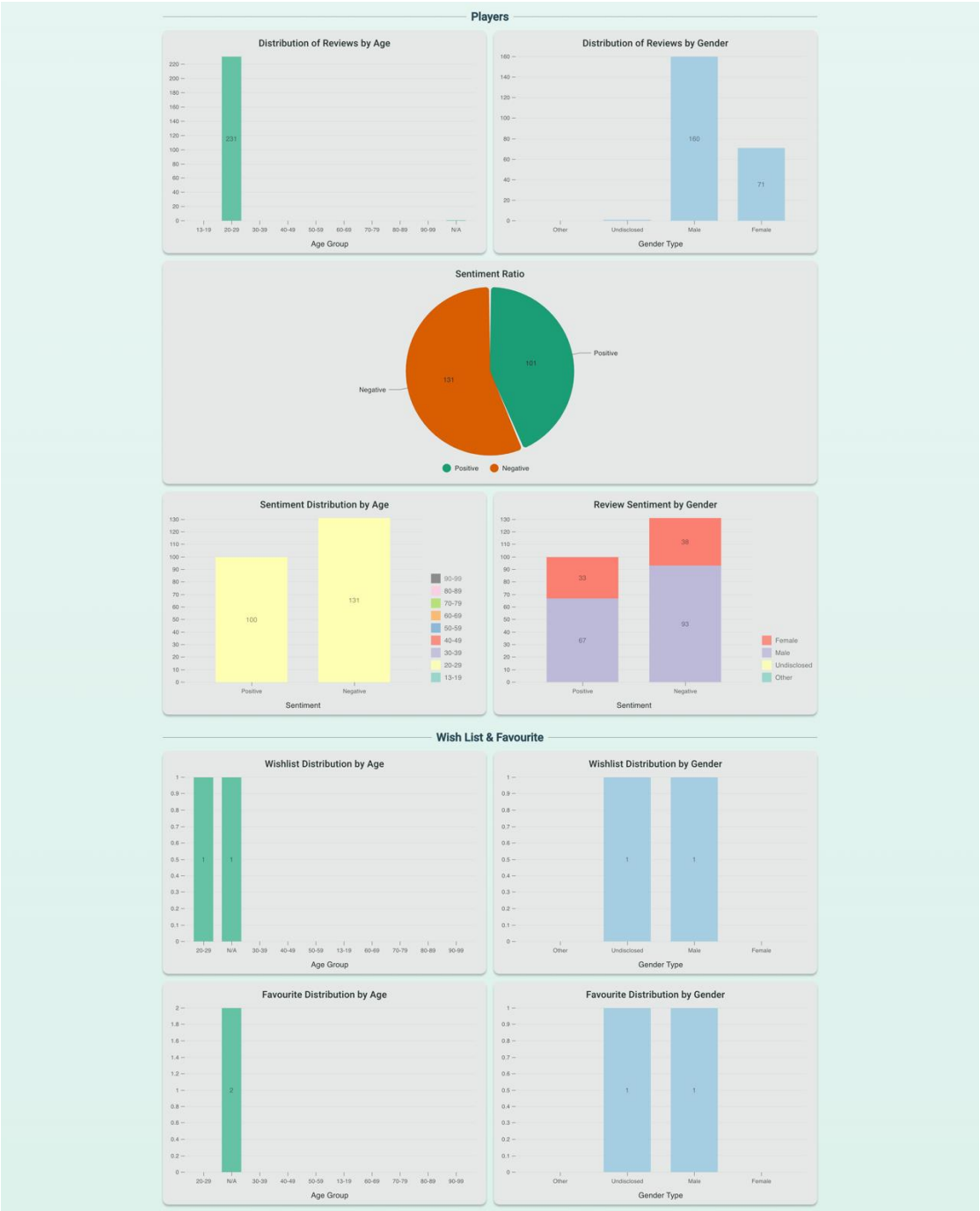



*Figure 81 Player Statistics and Wishlist & Favorite Statistics of the Game Starfield in the Analytics Page*

The analytics page is divided into 4 main sections, including Game Statistics, Review, Players and lastly Wishlist & Favorite, each having their graphs based on the information available on our platform and will be discussed individually.

- Game Statistics

  The game statistics section displays the number of users that favorited, wish listed and reviewed the game. In addition, the game's score (average of the reviews), percentile ranking, and recommendation ratio are also displayed as pie charts.

- Reviews

The review section shows the distribution of the reviews' length, player's platform, and playtime as input by the reviewers, allowing developers and players to understand the general segments of player reviews.

- Players

  The Players section displays the distribution of the age and gender of reviewed players and their respective sentiment distribution. Lastly, it also includes the overall sentiment ratio (distribution of positive and negative reviews).

- Wish List and Favorite

  The wish list and favorite section provide 4 bar charts based on the user's age and gender that wish listed and favorited the game on the game page, allowing the developers and players to understand their target players, including their age group and gender.

As the page is divided into 4 main sections, each containing many graphs, a Floating Action Button is available on the top left of the page, allowing users, particularly mobile users, to quickly view their current section and navigate to other sections if needed by clicking on the section labels as shown (See Figure 82). To reduce obstruction to the graphs, the FAB will be collapsed by default, improving the user experience.


*Figure 82 Floating Action Button that tracks the user's current section based on the viewport and allows for quick navigation*

### 4.5.9 Review Page

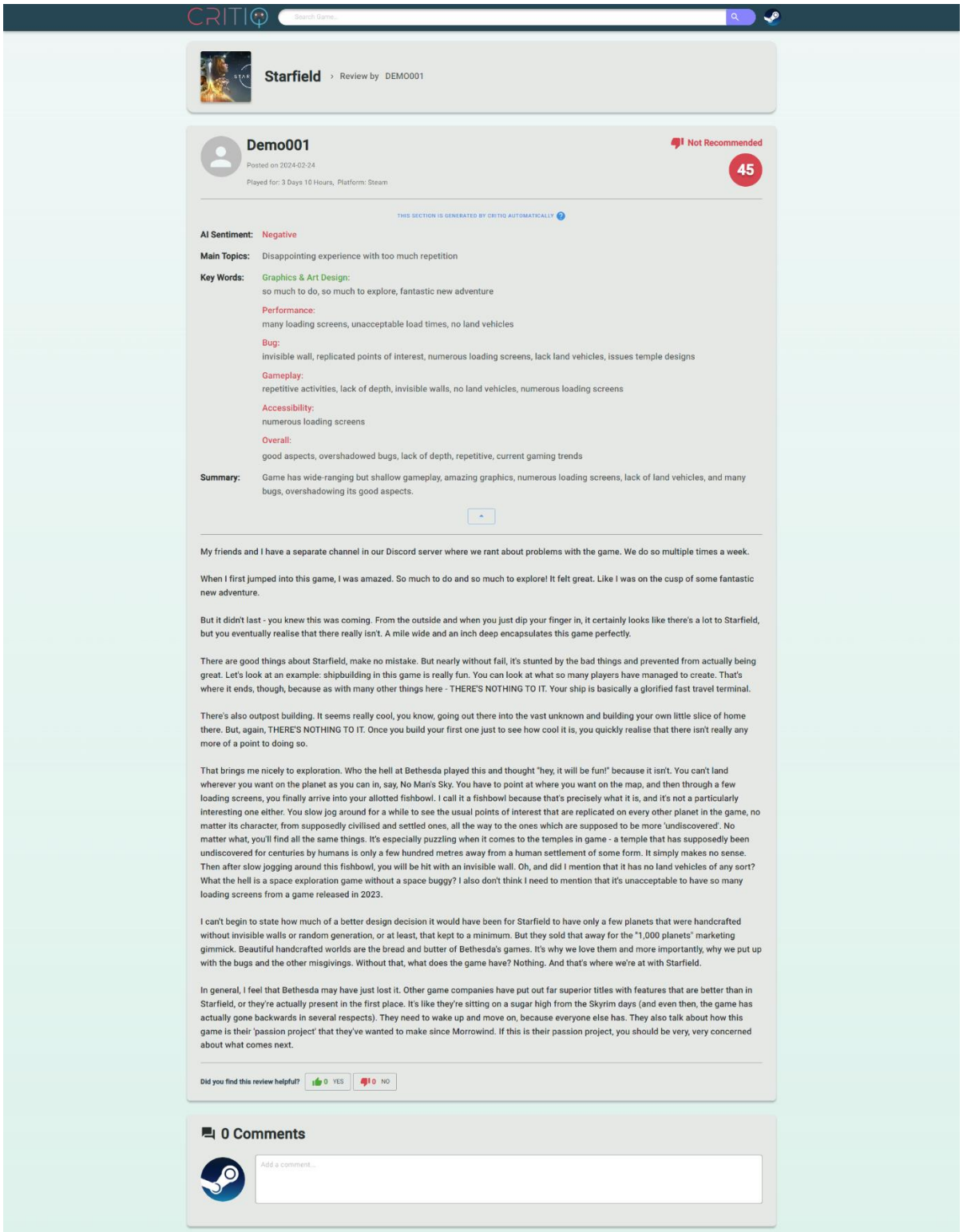



*Figure 83 Web application's review page design*

This section presents the review page, which was introduced in the previous section. The review page can be accessed by clicking on the read more button on the game review card and consists of three sections: the header section, the review body section, and the review comment section (See Figure 83).

The header section, located at the top of the page, displays the game icon and a location-based breadcrumb navigation. The breadcrumb navigation allows the user to return to the previous level in the website's hierarchy, which is the game page, by clicking on the game icon or the game name. The Breadcrumbs component from MUI is used to implement the breadcrumb navigation.

The review body section, which follows the header, comprises four sub-sections. The first sub-section displays some basic information about the review and the reviewer. On the left, it shows the avatar icon and the name of the reviewer, the creation time of the review, the platform, the total play time, and the version of the game that the reviewer played. On the right, it shows the recommendation status of the review and the score that the reviewer assigned to the game.

The second sub-section consists of the review analysis information. This information within the section is generated automatically using the results of the NLP models (See Figure 84), and the user can toggle this section by clicking on the up or down arrow button. All NLP-related solutions for review analysis have been implemented, including a binary Sentiment Analysis, Main Topic related to the review, Keywords based on the aspects sorted by the sentiments, and a summary of the review.

- Sentiment Analysis: The application will visually represent the sentiment of the review. If the review is positive, it will be highlighted in green. Conversely, if the review is negative, it will be highlighted in red. This immediate visual cue helps users quickly gauge the overall sentiment of the review.
- Main Topic Identification: The application will analyze the review and identify the main topic associated with the game's genre. It does this by comparing the content of the review against a database of the top 15 topics related to that genre. The identified main topic will then be prominently displayed, providing users with a quick understanding of the review's focus
- Aspect-Based Keyword Display: Keywords that pertain to various aspects of the game will be extracted and displayed if they are mentioned in the review. These aspects could include gameplay, graphics, sound, and more. The keywords will be organized to show positive aspects first, followed by any negative aspects. If certain

aspects are not mentioned in the review, they will not be displayed, keeping the interface clean and focused on relevant information.

- Review Summary Generation: For reviews that are more than 50 words in length, the application will generate a concise summary. This summary will capture the key points of the review, allowing users to quickly absorb the most pertinent information without reading the entire text.

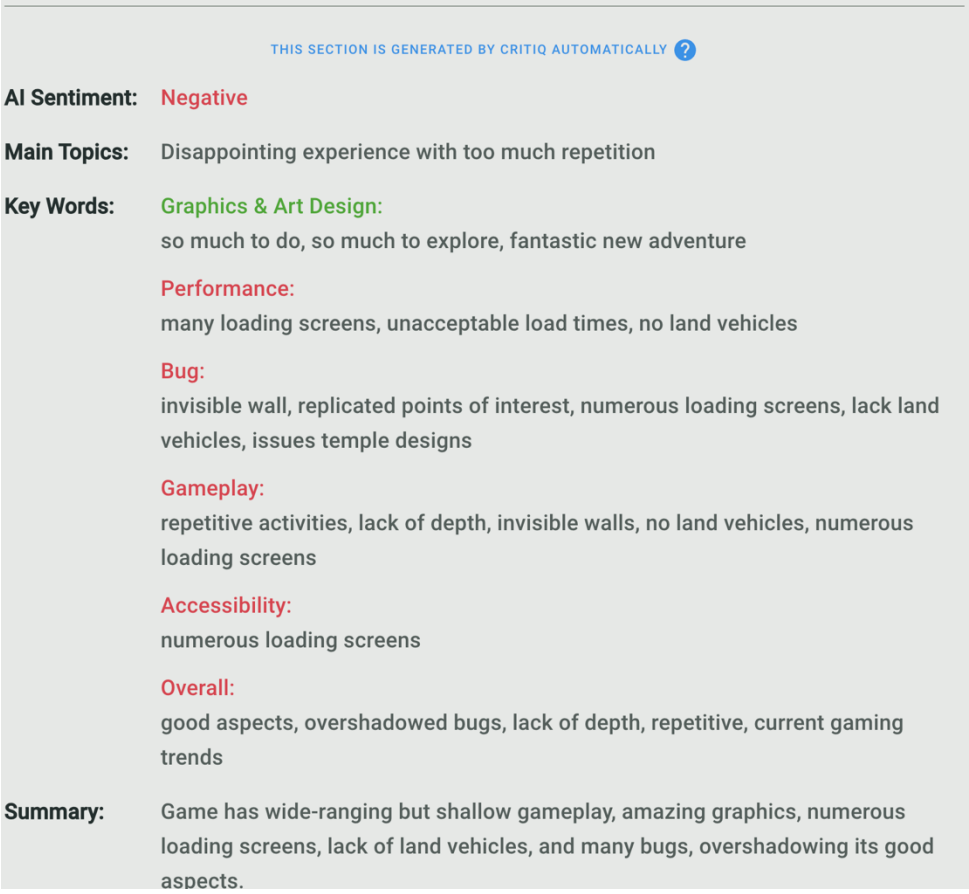



*Figure 84 Review Section that shows the NLP solutions, including Sentiment Analysis, Main Topic, Keywords, and Short Summary*

The third section presents the review context and attached images. The section displays the review text body and an image slider that shows the images associated with this review. The image slider is implemented using the Embla Carousel component, which offers advanced features such as a responsive layout based on viewport and thumbnail navigation. The height of the slider is reduced for the mobile viewport to prevent the slider from taking up the entire screen. The user can navigate the images in various ways, such as dragging on the image, clicking the left and right arrow buttons, and clicking on the individual thumbnail below the slider. The final section allows the user to evaluate the review by clicking on the like or dislike buttons, depending on whether they found the review helpful or not. The buttons and the game review card show the

number of likes and dislikes, which provide an initial impression of the review to other users.

The final section of the review page is the comment section, which appears below the review body section. This section allows the users to add new comments and interact with other users about the review. The section displays the total number of comments at the top, followed by the comment box, which is only visible to logged-in users. The comment box shows the user's avatar and a text input field for entering the comment. The user can submit the comment by pressing enter while typing. The comments of other users are shown below the comment box, sorted by the creation time in ascending order. The user can browse the comments using the pagination at the bottom, which is implemented in the same way as the search result page (see Section 4.2.3).

### 4.5.10 Progressive Web Application

Progressive Web Applications (PWAs) are a cutting-edge approach to software development that merges the advantages of traditional websites with those of native applications. Here's a more detailed breakdown of their features and benefits:

- Cross-Platform Compatibility: PWAs are built using standard web technologies such as HTML, CSS, and JavaScript, which allows them to operate across various operating systems and devices. This is possible because they run within web browsers, which act as a universal platform.
- Single Codebase: Developers can create a PWA once and deploy it everywhere, eliminating the need to develop separate versions for different platforms. This unified codebase approach simplifies maintenance and updates.
- Installation and Accessibility: Unlike conventional apps that require downloading from an app store, PWAs can be installed directly from a browser. They can be added to a device's home screen or desktop, providing quick access just like a native app. This feature is particularly beneficial in regions with limited internet connectivity or for users with restricted data plans. Most Modern Desktop and Mobile operating Systems, including iOS, Android, MacOS and Windows will treat the PWA installed as a standalone application, searchable and launchable with the OS's search feature (See Figure 86).

- Performance: PWAs are designed to be fast and responsive. They leverage modern web capabilities to deliver smooth animations and transitions, mimicking the feel of a native app.
- Shareability: Since PWAs are accessed via URLs, they can be easily shared through a simple link, which is much more straightforward than guiding someone to search for and install an app from a store.
- No App Store Dependencies: PWAs bypass app stores, allowing developers to update their apps at any time without waiting for store approvals. This leads to a faster update cycle and ensures that all users have the latest version.
- Security: PWAs are served through HTTPS, ensuring that the connection between the user and the application is secure and that the data exchanged is encrypted.

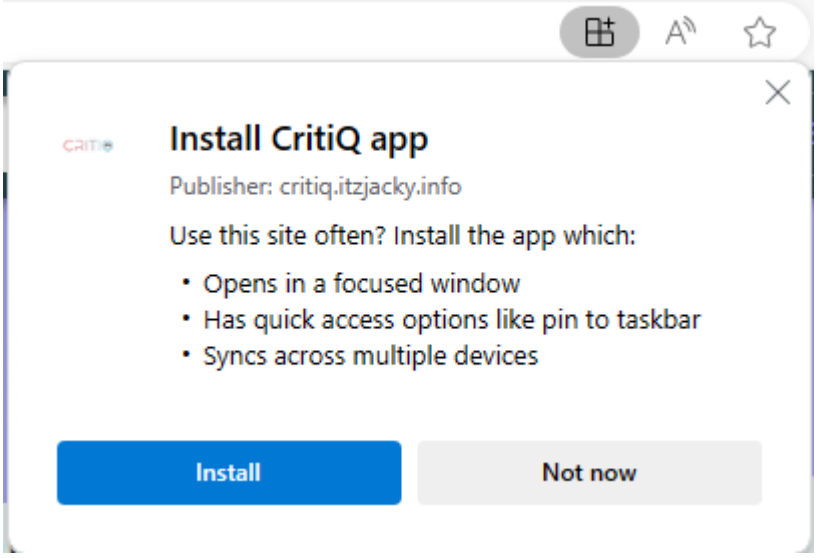



*Figure 85 PWA install button on the Chromium-based Microsoft Edge browser*

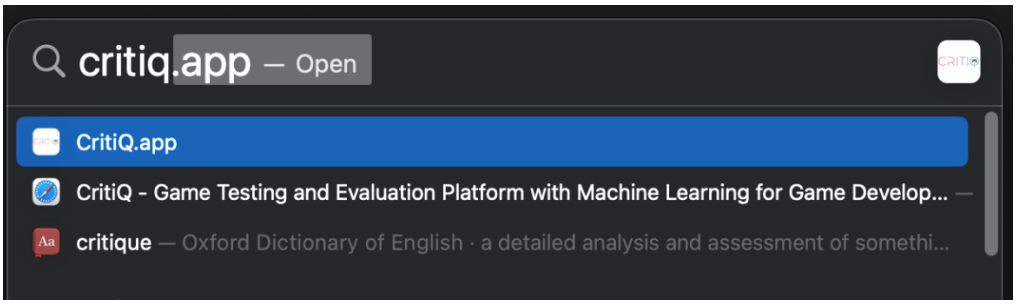



*Figure 86 CritiQ PWA searchable in MacOS Spotlight Search*

Our application leverages *next-pwa*, a zero-configuration package integrated with Next.JS, to streamline the adoption of PWAs without the complexities of manual worker insertion or manifest setup. Next-pwa automates the generation and registration of service workers, optimizing both precaching and runtime caching processes. The

only requirement is a simple modification to the `next-config.ts` file, making it effortless to integrate `next-pwa` into any modern Next.JS application.

The transition to PWA technology offers substantial benefits. It extends the reach and accessibility of applications by enabling them to function across diverse platforms such as iOS, Android, and desktops, all from a single codebase. This universality allows applications to be more readily available to a broader audience, as they can be conveniently placed as icons on home screens or desktops. PWAs also enhance user engagement through a seamless user experience and features like push notifications, which maintain user connection even during inactive periods. Moreover, PWAs are designed for speed, with service workers caching essential resources to expedite load times, ensuring that users aren't kept waiting for content to load. From a development standpoint, PWAs are cost-effective, as maintaining a single codebase is more economical than developing and updating separate native applications for different platforms. Any updates made to the main repository are immediately reflected in the PWA, simplifying the update process.

In essence, PWAs serve as a bridge between the capabilities of modern web technologies and the performance of native applications, delivering an experience that is both accessible and efficient, with a performance that closely rivals that of native apps.

### 4.6 Web Scraping

Web scraping was adopted to extract information of existing games on the Steam platform. Using Steam Web API (Valve, 2023), 157068 games in total were discovered, and data such as description, developers, genres, categories, and release dates were scraped from Steam in October 2023. The data was then saved in JavaScript Object Notation (JSON) format.

Subsequently, a Python program was used to parse the data and modified them to match the format used in the database (See Figure 87). The refined data served two primary purposes: firstly, to create a robust foundational dataset for our platform, and secondly, to support our research in the field of topic modeling. The latter involved conducting experiments with various computational methods to develop distinct topic models. These models were trained using the most prominent categories and genres derived from the Steam data, aiming to capture the essence of the gaming content.

In addition to the game information, a substantial dataset comprising approximately 4 million filtered reviews was also scraped from Steam. This dataset was specifically curated to serve as the training and testing material for our Topic Modeling machine learning model. The reviews were selected to ensure relevance and recency, providing a rich resource for the model to train on based on new review inputs. This extensive preparation was crucial in creating a machine learning model that is both robust and reliable in its topic classification capabilities.

| | | | | | |
|---|---|---|---|---|---|
| Valve | 0 | CS2 | Valve | 0 | 2023 |
| Testing | 0 | Testing | Testing | [NULL] | 2023-09-22 |
| [NULL] | 0 | Monster Hunter World | Capcom | [NULL] | July 2018 |
| CDProjekt Red | 0 | Cyberpunk 2077 | CDProjekt Red | [NULL] | 2020 |
| CDProjekt Red | 0 | Cyberpunk 2077 Phantom Liberty | CDProjekt Red | 0.5 | 2023 |
| Bathesda Game Studio | 0 | Starfield | Bathesda | [NULL] | 2023 |
| Valve | 0 | Dota 2 | Valve | [NULL] | 2013 |
| Twilight Sonata Studio | 0 | 大富翁少女 18DLC | Twilight Sonata Studio | [NULL] | 10 Jan, 2022 |
| Game Creator's School | 0 | The Brave vs Dragon | Game Creator's School | [NULL] | 8 Jul, 2022 |
| Smunty Studios | 0 | FourPlay Chess | Smunty Studios | [NULL] | 30 Apr, 2022 |
| DigitalDream | 0 | Scratch Girl | DigitalDream | [NULL] | 14 Dec, 2021 |
| Wackytoaster | 0 | Rimebeard Demo | Wackytoaster | [NULL] | 11 Nov, 2021 |
| E.144 | 0 | Yavi | E.144 | [NULL] | 13 Dec, 2021 |
| Quentin Edel | 0 | Find The Cat | Quentin Edel | [NULL] | 6 May, 2022 |
| Paweł Wiecha | 0 | Cosmos Conquer | Iguana Mercenary | [NULL] | 4 Mar, 2022 |
| | 0 | Oasis: Dark Forest Playtest | | [NULL] | 10 Nov, 2021 |
| Immure Creations® | 0 | The Infecting 3 | Immure Creations® | [NULL] | 30 Nov, 2021 |

*Figure 87 Sample Database Records of Scraped Games from Steam*

## 4.7 Backend System

This section will discuss the preliminary results of the backend solution, including CI/CD (Section 4.4.1), API Endpoints and Database (Section 4.4.2), API Security (Section 4.4.3), S3 Bucket (Section 4.4.4) and Stability and Testing (Section 4.4.5).

The backend infrastructure has been established in line with the proposed methodology, utilizing Spring Boot as the core framework. Continuous Integration and Continuous Deployment (CI/CD) processes are managed through Jenkins and Docker, as depicted in Figure 88.

To accommodate the increased computational demands and the variety of services, two virtual machines have been provisioned. Each virtual machine is configured with a dedicated CI/CD pipeline. This strategic setup is designed to enhance the efficiency of the build process and to reduce any potential service interruptions during application updates. The individualized pipelines ensure that each service can be updated independently, thereby streamlining the deployment process and maintaining a high level of availability for the applications.

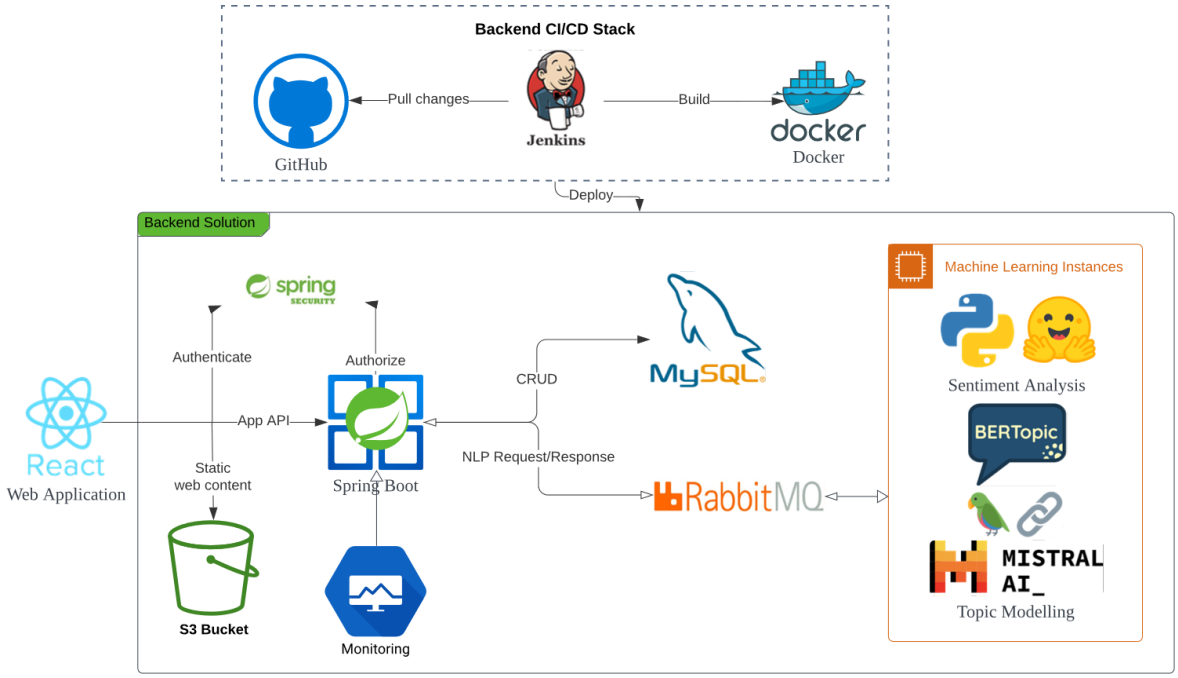



*Figure 88 Backend Solution Architecture Graph*

### 4.4.1 CI/CD

Uptime and Stability are key for modern web applications. And our backend architecture is designed to provide high uptime and stability without relying on additional backup nodes. Our pipeline is written to only deploy changes to the modified service, reducing the overall system downtime for long-starting services, including our different machine learning Model services.

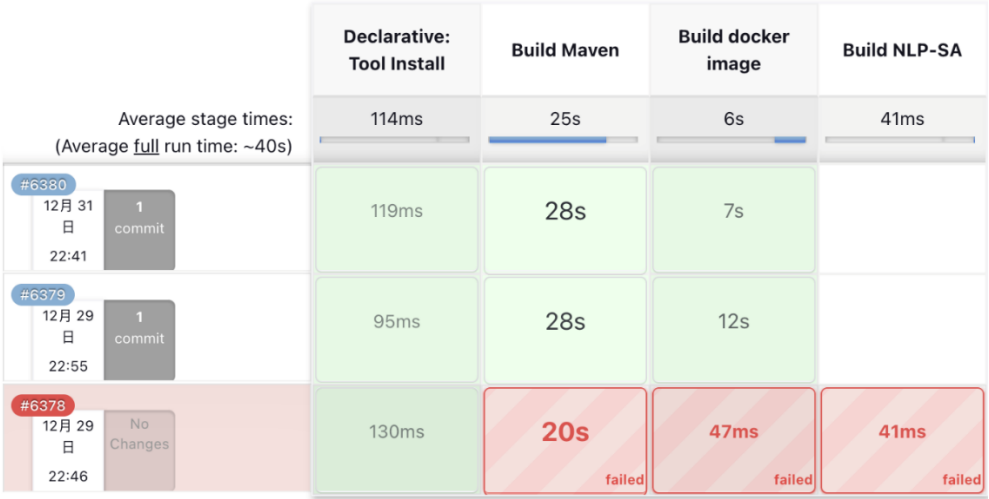



*Figure 89 Jenkins deployment User Interface with different stages of deployments.*

By using an efficient and custom CI/CD pipeline (See Figure 89), we have reduced the downtime for our backend system during deployment to approximately 10 seconds with minimal impact on the end-user experience of our platform. Additionally, Jenkins provides a user-friendly interface that clearly presents the deployment status and timelines, with any unsuccessful deployments highlighted in red for immediate visibility.

In the event that any bugs or issues escape detection during the testing phase and subsequently cause a failure during deployment, the stability of the production environment remains unaffected. This is due to the safeguard in place where a failed build will exit the pipeline and the production container is only updated with a new container after a successful build. Therefore, the existing live version continues to operate without interruption, ensuring uninterrupted service until the new deployment is verified and builds successfully.

The creation of Jenkinsfile and Dockerfiles streamlines the deployment process, enabling the encapsulation of our applications into containers. These containers represent discrete, manageable units that can be swiftly deployed or halted via the Jenkins pipeline, offering a high degree of control and efficiency in managing application lifecycles.

### 4.4.2 Message Queue Implementation

In the system architecture (see Section 3.3.7), RabbitMQ serves as the backbone for inter-process communication, bridging the Spring Boot Application with the machine learning Python instances. This setup is pivotal in achieving the desired scalability and fault tolerance. To optimize this, distinct input and output queues are designated for each principal functionality—Sentiment Analysis, Topic Modelling (including Keyword Extraction), and Game Aggregated Review. This segregation ensures that I/O congestion is mitigated, allowing for efficient data flow.

Additionally, a dedicated **TestQueue** has been established. Its primary function is to log the inputs and outputs from the other queues, providing a robust mechanism for debugging and performance logging. This queue acts as a critical tool for developers to monitor and troubleshoot the system. Embracing a microservices architecture, the system ensures that each core functionality is managed by a singular microservice, or a Python instance, encapsulated within its own Docker container. This approach not only simplifies the management of services but also enhances the isolation and independence of each service, contributing to the overall resilience of the system (see Figure 90).

| Overview | | | | | Messages | | | Message rates | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Virtual host | Name | Type | Features | State | Ready | Unacked | Total | incoming | deliver / get | ack |
| / | GameAggregationQueue | classic | D Args | idle | 0 | 0 | 0 | 0.00/s | 0.00/s | 0.00/s |
| / | GameAggregationResultQueue | classic | D Args | idle | 0 | 0 | 0 | 0.00/s | 0.00/s | 0.00/s |
| / | SentimentAnalysisQueue | classic | D | idle | 0 | 0 | 0 | 0.00/s | 0.00/s | 0.00/s |
| / | SentimentAnalysisResultQueue | classic | D | idle | 0 | 0 | 0 | 0.00/s | 0.00/s | 0.00/s |
| / | SentimentAnalysisResultTestQueue | classic | D Args | idle | 0 | 0 | 0 | 0.00/s | 0.00/s | 0.00/s |
| / | TestQueue | classic | D | idle | 61 | 0 | 61 | 0.00/s | 0.00/s | 0.00/s |
| / | TopicModelingQueue | classic | D Args | idle | 0 | 0 | 0 | 0.00/s | 0.00/s | 0.00/s |
| / | TopicModelingResultQueue | classic | D Args | idle | 0 | 0 | 0 | 0.00/s | 0.00/s | 0.00/s |

*Figure 90 RabbitMQ user interface showing all of the created queues*

To augment tracking capabilities, the system meticulously records additional metrics such as token usage by LangChain. These metrics are stored in the database for analytical and billing purposes. An example of the data structure used to capture this information is shown in Figure 90, which presents a JSON object detailing the

breakdown of token usage across different categories within the message queue's output response (see Figure 91).


```
{
 "tokenUsageBreakdown": {"total_tokens": 2275, "spam_tokens": 650, "aspect_response_tokens": 975, "keywords_tokens": 325}
}
```


*Figure 91 sample JSON output for LLM token usage in message queue output response*

Ensuring message integrity is paramount. The system is designed to discard any message that fails to commit completely. Similarly, messages that are not fully acknowledged due to processing failures are not consumed. This strategy is crucial in maintaining the durability and reliability of the message queue, guaranteeing that only complete and successful transactions are persisted.

The architecture's scalability is one of its most significant features, thanks to the implementation of RabbitMQ. When there is an increase in user demand or an unexpected spike in traffic, the system can be easily tuned to respond flexibly and spin up additional Python instances. These instances can be dynamically created and integrated into the existing workflow without disrupting ongoing processes with minimal changes to the codebase. This is possible because each instance operates independently, consuming messages from the designated queues as they become available.

Moreover, the use of Docker containers for each Python instance further streamlines this process. Containers can be rapidly deployed, with each new instance being an exact replica of the existing ones, ensuring consistency and reliability when connected to the message queue broker. This elasticity allows the system to maintain high performance and availability, even during peak usage times, providing a seamless experience for users. Scalability and fault tolerance are thus inherently built into the system, ready to accommodate growth and fluctuations in demand with ease.

### 4.4.3 API Endpoints and Database

The APIs required by the frontend application have been developed and tested to optimize for performance and stability.

By creating API endpoints based on the need for frontend applications and optimizing database performance, the Round-Trip-Time (RTT) of the most commonly used API endpoints has been lowered to within 300ms.

Optimization strategies have been applied to our database to enhance the application's performance. By implementing indices on entities that are frequently queried, we've managed to decrease the round-trip time (RTT) for most API endpoints. This has led to a noticeable improvement in the application's responsiveness. Additionally, we've adopted a lazy loading approach, which defers the loading of information until it's specifically required by an associated entity. This efficient data handling technique contributes to the overall speed and efficiency of the application. A prime example of these optimizations in action is the /findGameById API endpoint. It's commonly used to retrieve details about a particular game and, thanks to our enhancements, this operation now boasts a swift execution time of just 65 milliseconds (See Figure 92).

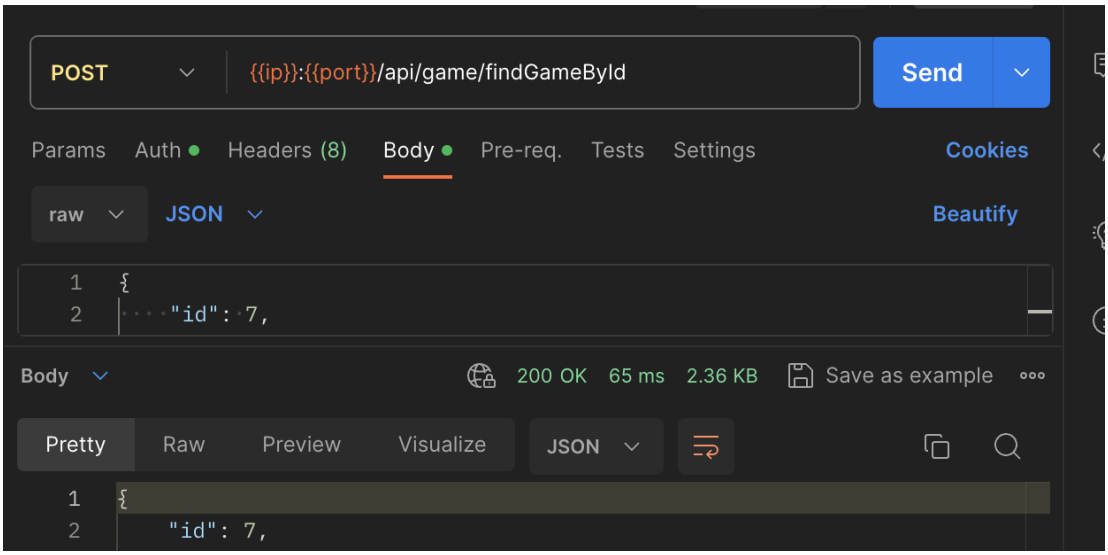



*Figure 92 API call to /findGameById to fetch specific game information finishes in 65ms*

For API calls that perform exhaustive searches, including the Advanced Search feature, the RTT depends on the size of the returned result. Testing has shown that the worst case's RTT still falls below 300ms (See Figure 93).

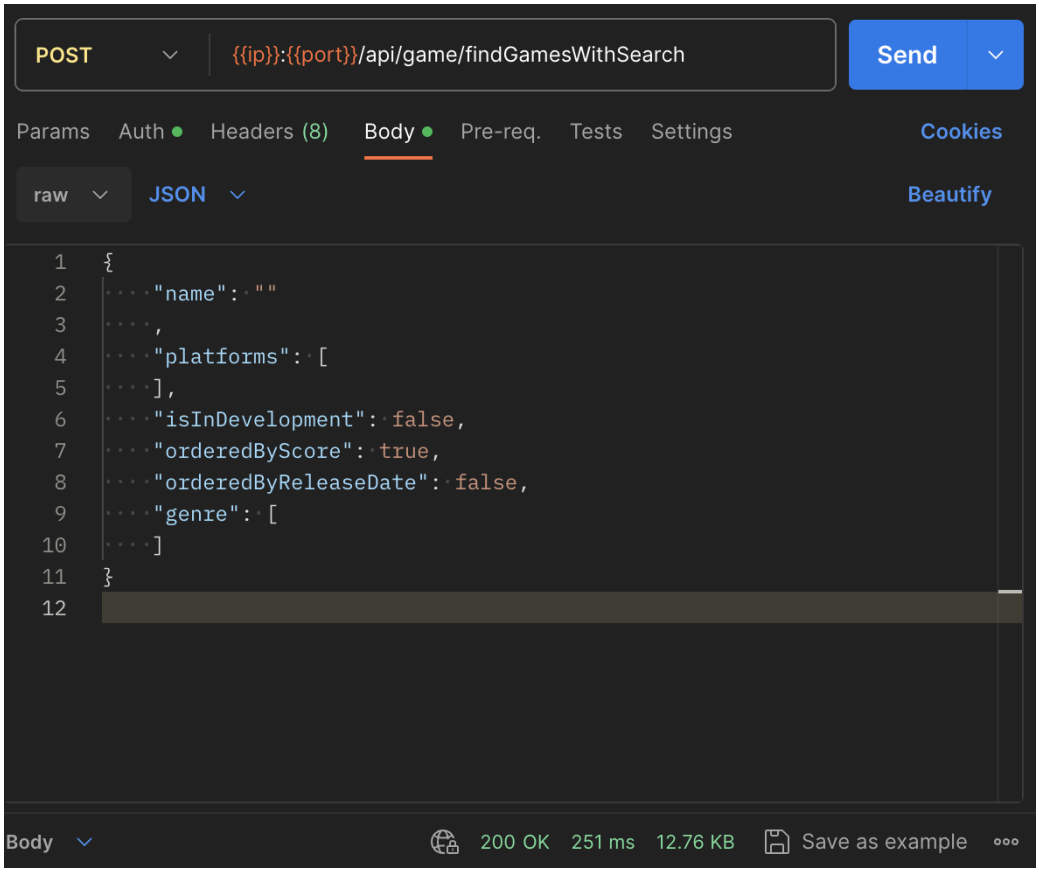



*Figure 93 API call to /findGamesWithSearch to perform exhaustive game search finishes in 251ms*

Our database plan offers a connection limit of 150 concurrent connections from any location (See Figure 94). To improve the performance of complex queries, including Advanced Game Search or Data Analysis, setting a larger connection size from Spring Boot improved the query times by over 100%. Further testing has shown that 70-100 concurrent connections to the database in the production environment yielded the best result without utilizing too many connections (See Figure 95). A small portion of the connection pool is reserved for local development and testing purposes. To further optimize database performance, all common queries on User, Game and Reviews have proper indexing using a B+-tree to improve common range and selection queries. In addition, many optimizations were also performed using Spring Data JPA and Hibernate, including imposing Read-only Transaction when necessary, the use of Lazy Loading to prevent loading unnecessary related entities and Batch Fetching to reduce the number of individual database queries performed, minimizing the N+1 query problem commonly found in Largescale database queries, and the use of paging and

sorting together through defining the size of each page, the criteria for sorting and the number of desired page in a particular order to reduce data processing time.

1 vCPU / 2 GB RAM / Storage minimum: 30 GB / Connection limit: 150

*Figure 94 Current Database Plan with 2GB RAM offers a maximum of 150 concurrent connections*

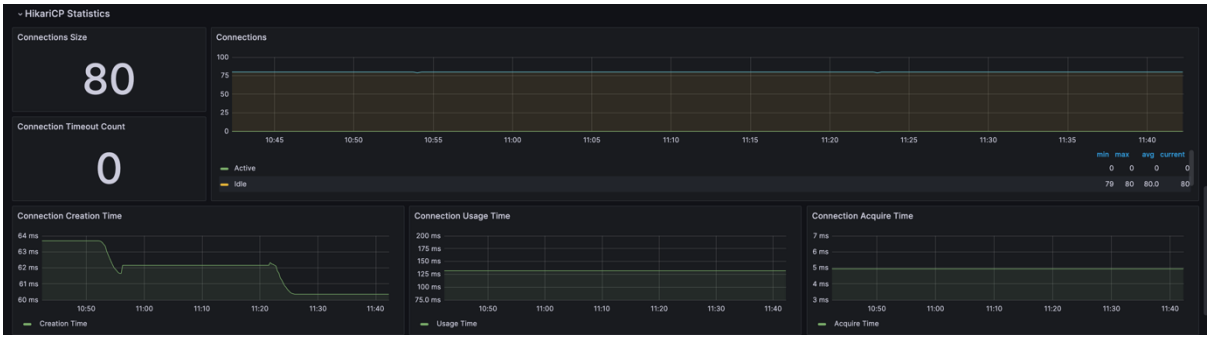



*Figure 95 Spring Boot utilizes 80 concurrent connections to the database*

In the pursuit of optimizing database performance, several strategies were employed. One of the key strategies was the implementation of indexing using a B+-tree for all common queries on User, Game, and Reviews. This technique significantly improved the efficiency of common range and selection queries. A B+-tree, a self-balancing tree data structure, maintains sorted data and allows for efficient insertion, deletion, and search operations. It is particularly beneficial for systems with large amounts of data and wide keys, such as databases and file systems. In addition to B+-tree indexing, the project also utilized the capabilities of Spring Data JPA and Hibernate, which are Java frameworks that simplify the implementation of data access layers. These frameworks were used to perform several optimizations. One such optimization was imposing Read-only Transactions when necessary. A read-only transaction, where data is only read and not modified, can improve performance as the database doesn't have to log the transaction or perform any concurrency control mechanisms, which are required for transactions that modify data.

Another optimization was the use of Lazy Loading, a design pattern that delays the loading of related entities until they are specifically requested. This prevents the loading of unnecessary related entities, thereby improving the efficiency of the system. Furthermore, Batch Fetching was employed to reduce the number of individual database queries performed, minimizing the N+1 query problem commonly found in large-scale database queries.

Lastly, the project implemented paging and sorting through defining the size of each page, the criteria for sorting, and the number of the desired page in a particular order. This technique reduced data processing time, further optimizing the system's performance. These strategies collectively contributed to the creation of a robust and efficient system, demonstrating the potential of these techniques in optimizing database performance.

### 4.4.4 Authentication

Application Security is the linchpin in preserving the confidentiality, integrity, and availability of our platform's data, ensuring that access and modifications are strictly governed. The implementation of Spring Security with JWT (JSON Web Tokens) is a critical component of our security infrastructure. This robust framework employs the HS256 Signature Algorithm that encrypts the Sign-In key using a centralized server password and a BCrypt password encoder, a highly secure encryption method that hashes user passwords to protect against unauthorized access. In tandem with this, the system generates unique UUID tokens for each session, which not only bolsters security through unpredictability but also aids in tracking and managing user sessions.

Token management is handled with precision, where tokens are programmed to expire after a pre-defined period, necessitating their timely refresh to maintain continuous access. This mechanism is crucial in preventing the prolonged use of stale credentials, thereby reducing the risk of security breaches.

The JWT filter plays a pivotal role in the security apparatus, meticulously extracting claims from tokens and generating new JWTs as needed. It ensures that each token is imbued with the correct user claims, which are essential for authorizing user actions within the platform. The configuration of CORS is another vital aspect of our security measures. It is carefully set up to allow requests from authorized origins while blocking potentially harmful cross-origin interactions, thus safeguarding the API from certain types of web-based attacks.

Our security model is further strengthened by a clearly defined role hierarchy, which delineates the access levels and permissions associated with different user roles. This hierarchical structure is instrumental in enforcing the principle of least privilege,

ensuring that users can interact with the system within the bounds of their necessary functions.

At the core of our security architecture is the security chain, a comprehensive filter that scrutinizes every incoming request. This chain is the final arbiter of security, validating each request against our authentication and authorization criteria. It is the bulwark that stands guard, processing requests to ensure that they comply with our security protocols before any operation on the data is performed.

For our application, two types of tokens were implemented: the access token and the refresh token. The access token is akin to a digital key that grants temporary, scoped access to the user's resources or session. It is typically short-lived, expiring after a brief period to mitigate the risk of unauthorized use if intercepted. This token is used with each API request, allowing the server to validate the user's identity and permissions without requiring them to re-enter their credentials. The refresh token, on the other hand, serves a different purpose. It is used to obtain a new access token when the current one expires. Refresh tokens have a longer lifespan and are securely stored on the client side. When an access token is nearing expiration, the client can present the refresh token to the server to issue a new access token, thus maintaining the user's session without asking for their credentials again. This process ensures a seamless user experience, as it allows users to stay authenticated over longer periods of inactivity.

Together, these tokens provide a secure and efficient way to manage user sessions and access control in modern web applications. The access token ensures that requests are authenticated and authorized for the duration of its validity, while the refresh token minimizes the need for frequent re-authentication, enhancing security and user convenience.

In essence, our security strategy is a multi-faceted approach that integrates Spring Security with JWT, BCrypt password encoding, UUID token generation, cautious token management, claim extraction, CORS setup, role hierarchy, and a robust security chain. Together, these elements form an impregnable fortress, safeguarding our platform and its data against unauthorized access and ensuring that our users can trust the security of their interactions with our system.

### 4.4.5 API Security

API Security is crucial in maintaining the confidentiality, integrity, and availability of our platform and its data by only permitting data access and modifications.The system uses the **@AuthenticationPrinciple** annotation in the REST API endpoint of the backend application to obtain the user based on the username or email from the JWT in the HTTP request. The system can also restrict access to the API call by verifying the user's information. For instance, as demonstrated (See Figure 96), the system verifies that the user ID within the JWT matches the ID of the user requesting a verification email before proceeding with the operation.

```
@PostMapping("/sendVerifyEmail")
public ResponseEntity<Void> sendVerifyEmail(@RequestBody UserRequest userRequest, @AuthenticationPrincipal User u){
    if (u == null || !Objects.equals(u.getId(), userRequest.getId())) {
        throw new AccessDeniedException("Access Denied");
    }
```

*Figure 96 Access Control by verifying the user based on the JWT sent in HTTP requests using the @AuthenticationPrincipal annotation.*

In addition to @**AuthenticationPrinciple**, we utilize the @**PreAuthorize** annotation, which leverages the Spring Expression Language (SPEL) to authorize users directly from the JWT token. This powerful annotation evaluates permissions in real-time based on SPEL expressions. Within our application, @**PreAuthorize** is crucial for enforcing role-based access control. It ensures that certain API endpoints, like /**removeGame**, are exclusively accessible to users with the ADMIN role. Should a non-ADMIN user attempt to access this endpoint, the system is configured to reject the request and return a 403 Forbidden Error (See Figure 97). This layered approach to security effectively maintains the confidentiality, integrity, and availability of our platform's data and services.

```
@PreAuthorize("hasAuthority('ROLE_ADMIN')")
@PostMapping("/removeGame")
public ResponseEntity<Void> removeGame(@RequestBody GameRequest gameRequest, @AuthenticationPrincipal User u) {
    try {
        gameService.removeGame(gameRequest);
        return ResponseEntity.noContent().build();
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatusCode.valueOf( code: 400), e.getMessage());
    }
}
```

*Figure 97 Access Control by verifying the user's role based on JWT sent in HTTP requests before method invocation using the @PreAuthorize annotation.*

Accessing a protected API endpoint requires proper authentication and authorization. If a request is made without a valid JWT token, or if the requester is not the authorized user or lacks the appropriate permissions, the system will deny access. This is enforced by returning a 403 Forbidden error, indicating that the server understands the request but refuses to authorize it. This safeguard is crucial for maintaining the security and integrity of the API and is visually represented (See Figure 98).

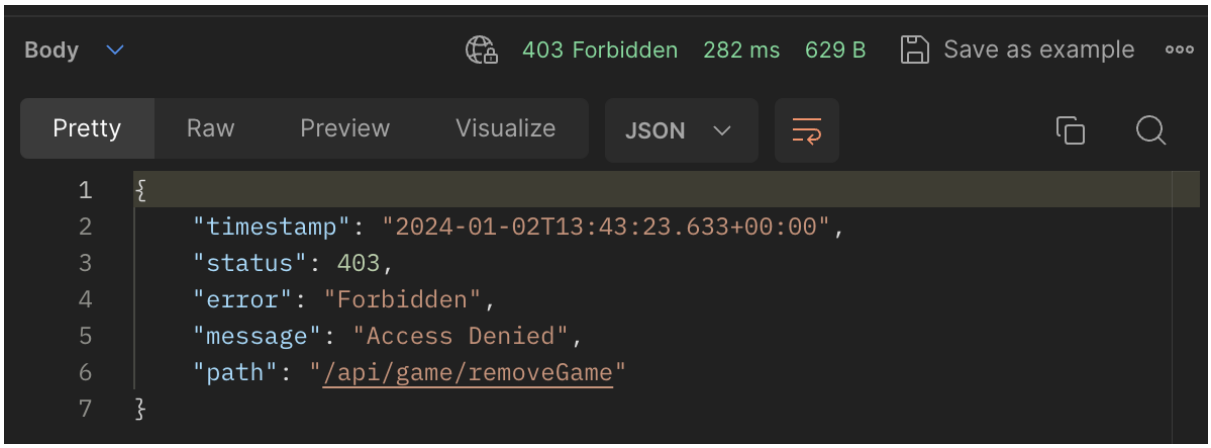



*Figure 98 403 Forbidden Error on Unauthorized Access to Protected API endpoints*

### 4.4.6 S3 Bucket Storage

The S3-compatible storage Solution provided by Digital Ocean offers high availability, To optimize the user's experience, a Content Delivery Network (CDN), is a network of edge servers that serve the content to the user based on the user's geographic location to minimize network traffic time, provided by Digital Ocean is used. By using a CDN, we can minimize the data fetching time of common User Interface elements in the web application, including User Icon, Game Images, and Review Images regardless of the user's geolocation as the contents are distributed across the globe in an edge network. Furthermore, by imposing write restrictions such as image modification on game reviews, the chance of a cache hit is increased and the latency for CDN synchronization minimized to optimize the user's experience in loading the objects.

Together with browser and server caches, images presented on the web application can be loaded efficiently and quickly without hindering the user experience during page navigation or exploration (See Figure 99).

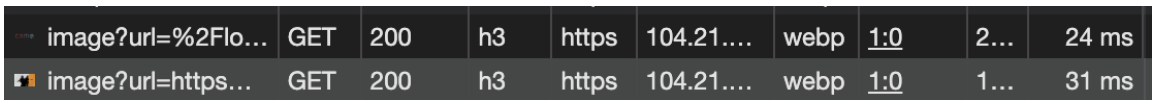



*Figure 99 Fetching of Cached Image(s) can be performed within 50ms*

To ensure the integrity of files uploaded to the storage, any modification to the storage bucket is only permitted through Spring Boot secured API endpoints, preventing unwanted access and modification of meta-data, including uploader, upload time, and data. In addition, additional information including the uploader of the files will be tracked during file upload for security purposes and tracking (See Figure 100) while the uploader for files uploaded by the Backend system will be marked as “System”.

```
metadata.setHeader("uploader", uploader);
```

*Figure 100 Uploader Header is set to the user's name during file upload*

### 4.4.7 Stability and Testing

Gatling, with its powerful and flexible load-testing capabilities, played a crucial role in our API testing strategy. It allowed us to simulate a variety of user behaviors and traffic patterns to assess how our system would perform under real-world conditions. We meticulously crafted Gatling scenarios that mirrored a wide spectrum of user interactions, ranging from simple read operations to complex transactions involving multiple CRUD operations.

The tests were designed to incrementally increase the number of virtual users, thereby ramping up the load on our system to observe how it coped with escalating demands. Gatling's detailed metrics and reporting tools provided us with granular insights into the response times, request rates, and error rates for each type of operation. This data was invaluable in identifying bottlenecks and performance thresholds (see Figure 101). Moreover, Gatling's assertion feature allowed us to define specific criteria for acceptable system behavior, such as maximum response times and minimum successful request percentages. By setting these benchmarks, we could automatically verify that our system met the performance standards we had established.

Throughout the testing process, Gatling's real-time monitoring enabled us to observe the effects of the load on our system as it happened. This immediate feedback loop was instrumental in making on-the-fly adjustments to our test parameters and infrastructure configuration. The resilience of our system, as evidenced by the ability to handle 60 concurrent users with minimal impact on performance, is a testament to the effectiveness of Gatling in pushing our system to its limits. The average response time of 0.2 seconds across database queries, even under peak load, indicates that our backend infrastructure is not only robust but also optimized for high efficiency.

In conclusion, Gatling served not just as a load-testing tool but as a comprehensive performance evaluation framework that allowed us to validate the stability and scalability of our system. Its extensive features and detailed analytics played a pivotal role in ensuring that our application could deliver a consistent and reliable user experience, even under the stress of high traffic conditions. The insights gained from Gatling's testing have been instrumental in fine-tuning our system to handle the demands of our growing user base.

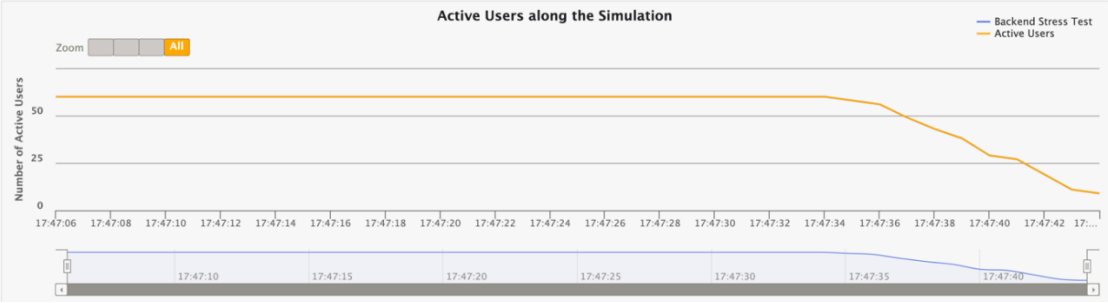



*Figure 101 Backend Load-Testing using Gatling, showing the ability to sustain 60 active users performing complex queries.*

# 5 Difficulties Encountered

This section discusses the main difficulties encountered, including HTTPS and SSL certificates (Section 5.1), Insufficient Computational Capacity (Section 5.2), and Message Queue Disconnection (Section 5.3).

## 5.1 HTTPS and SSL Certificate

The backend server, hosted on a Virtual Private Server (VPS) provided by Contabo, lacked an included Secure Sockets Layer (SSL) certificate, necessitating additional expenditure for its acquisition. This posed a significant challenge as modern web browsers, which prioritize security, default to converting HTTP requests to HTTPS, leading to a '404 Not Found' error when directed to an HTTP-only server. Consequently, the backend server was rendered incapable of receiving requests from the frontend application. To align with contemporary security protocols, the frontend application was deployed on Vercel, a frontend cloud-based hosting service that automatically provisions SSL certificates through Let's Encrypt, a widely-recognized open Certificate Authority (CA) from the Internet Security Research Group (ISRG). Let's Encrypt was created to enhance the security and privacy of the internet by promoting the widespread adoption of HTTPS which provides X.509 certificates for Transport Layer Security (TLS) encryption at no charge, aiming to make encrypted connections ubiquitous across the web. The initiative simplifies the process of setting up and maintaining TLS encryption, removing barriers such as cost, complexity, and manual certificate renewal, thus enabling a more secure and privacy-respecting web.

Implementing a self-signed SSL certificate can often lead to trust issues with mainstream web browsers such as Google Chrome, Firefox, and Safari. These browsers are designed to seek validation from trusted Certificate Authorities (CAs) to ensure the authenticity of a website's SSL certificate. When a certificate is self-signed, it lacks this third-party verification, signaling to the browser that it might not be secure. As a result, browsers will typically display a warning message to users, indicating that the site's security cannot be verified. This warning can deter users from accessing the site, as it raises concerns about the potential for data interception or manipulation. The error message, often accompanied by a visual cue such as a red lock icon or a strike-through on the 'https' in the address bar, serves as a caution to users that the connection is not private and that proceeding could expose them to risks. While self-signed certificates can encrypt data, they do not provide the assurance of a secure origin, which is a critical aspect of web security that users and browsers rely on. (see Figure 102).

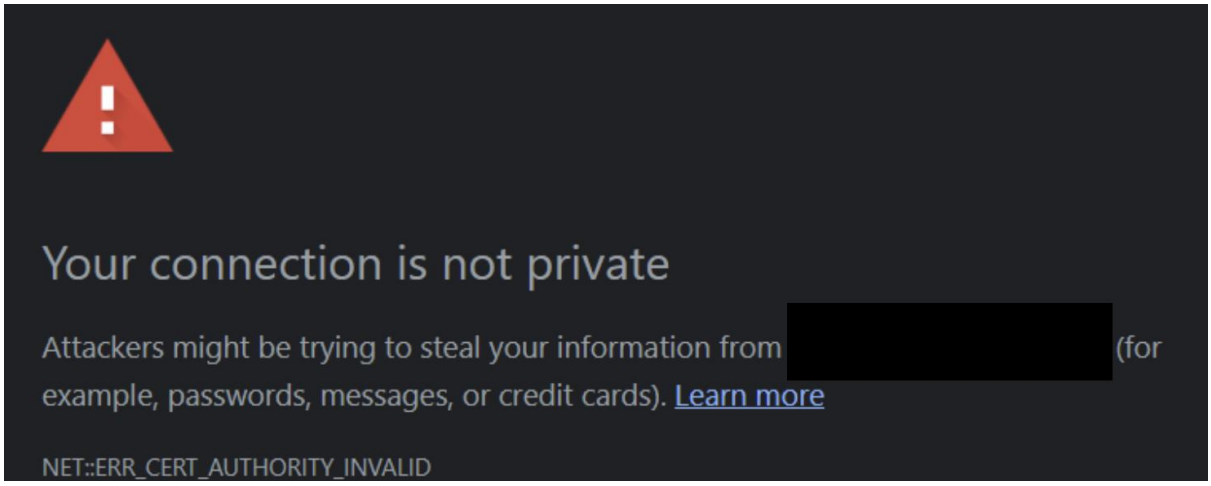



*Figure 102 Google Chrome Browser Error Page on Visiting Website with a self-signed digital certificate*

Our solution was to install Let's Encrypt locally on the VPS using CertBot to set up the SSL certificate to enable HTTPS for the traffic going to the Spring Boot backend services. Certbot is an open-source software tool that automates the process of obtaining, installing, and renewing SSL/TLS certificates from Let's Encrypt. It simplifies the task of enabling HTTPS on a web server, which is crucial for secure communication over the internet. Certbot is developed by the Electronic Frontier Foundation (EFF) to support many web services and operating systems and help streamline the management process of handling certificate issuance and renewal.

Let's Encrypt performs domain validation using challenges that only the domain owner can perform, making the certificate official compared to a self-signed certificate where no domain validation is performed. We faced some challenges when using the signed certificate file to enable HTTPS on Spring Boot. Firstly, the certificate generated was either in the PEM or Java Keystore (JKS) format, which were both incompatible with Spring Boot. We had to convert the keystore file into a PCKS12 format, which could be read by Spring Boot. Secondly, enabling HTTPS on Spring Boot required the front end to use HTTPS calls even in the local development environment by default. However, Spring Boot would not accept the certificate that was signed on the VM when it was run on localhost. To overcome this and facilitate local development, we disabled HTTPS during local development with different environment variables and allowed HTTP requests to be made to the local backend service.

It is pertinent to note that Let's Encrypt certificates have a validity period of 3 months, requiring manual renewal in the absence of an automated renewal service. The certificate

details, including the key, can be inspected using a modern browser when accessing the backend service (see Figure 103).

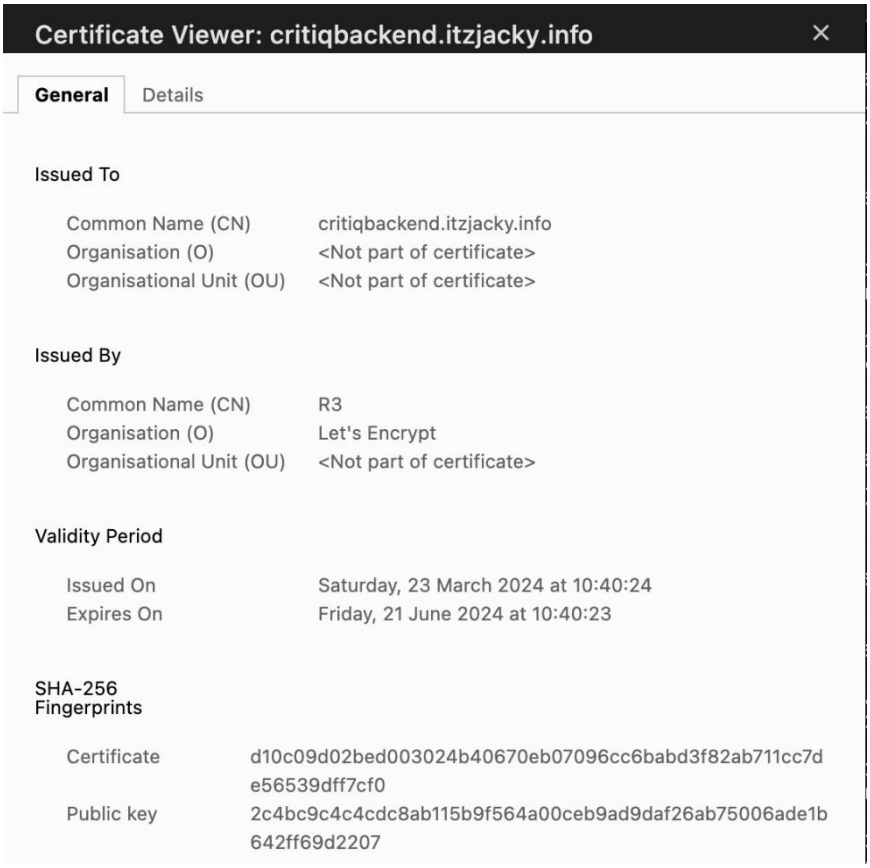



*Figure 103 Certificate Viewer on Backend Domain Address using the Google Chrome Browser*

### 5.2 Insufficient Computational Capacity

In the initial design phase, the backend architecture incorporated both the Spring Boot Web Server and the Sentiment Analysis Model on a singular Virtual Machine (VM). This VM was equipped with 4 virtual CPU cores and 8GB of RAM. Post-deployment of updated models and locally-ran LLM, it was observed that the VM's resource utilization was consistently high, with memory and CPU usage frequently surging to over 95% during periods of increased application activity. This excessive consumption of resources resulted in a marked degradation of performance, manifesting as prolonged response times within the web application and extended duration for the sentiment analysis model to process and classify data. The resource constraints of the VM were a bottleneck, adversely affecting the overall efficiency and user experience of the application.

In response to the resource limitations encountered with the original Virtual Machine (VM), a strategic upgrade was implemented, resulting in the establishment of a new VM boasting 16 virtual CPU cores and 64GB of RAM (see Figure 104). This enhancement was primarily

aimed at mitigating the high memory utilization experienced by the initial VM. Consequently, the Sentiment Analysis Model, along with the newly integrated Topic Modeling model, were transitioned to this upgraded VM. The selection of a machine with superior computational capabilities was deliberate, designed to expedite inference times by effectively addressing the memory swap bottleneck.

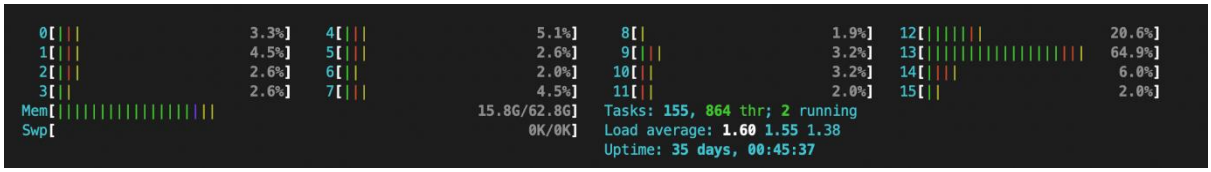



*Figure 104 10% CPU and 25% Memory usage for Machine Learning Models in new Virtual Machine*

Despite these improvements, subsequent evaluations revealed that the performance of the 7B Large Language Models (LLMs), formatted and quantized in GGUF for CPU-exclusive inference, was suboptimal, with the runtime for processing each review approaching an untenable 20 minutes. Efforts to refine the prompts yielded only marginal reductions in runtime. The absence of a CUDA-compatible GPU further exacerbated the situation, rendering the operation of even a modestly sized 2 Billion parameter LLM both arduous and time-intensive.

The ultimate resolution involved leveraging Mistral AI's official API, integrated with Langchain, to offload computational tasks to their external GPU infrastructure. This strategic shift resulted in a dramatic reduction in inference time to approximately 10 seconds per review (inclusive of network latency), representing a hundredfold improvement over local model execution. Furthermore, this approach enabled the utilization of the more robust Mistral 8x7B model, which, in comparison to the Ollama2 7B model run locally, delivered enhanced accuracy and a lower incidence of erroneous outputs, commonly referred to as 'hallucinations'. In the pursuit of balancing runtime efficiency and analytical precision, the decision was made to employ the smaller 8x7B model over the larger and more precise small/medium/large models. This choice was informed by considerations of both inference speed and cost-effectiveness, with the API expenses estimated at approximately 0.02-0.04 HKD per review processed. The implementation of this solution not only optimized performance but also ensured adherence to budgetary constraints.

Compared with existing GPU infrastructures offered by major cloud providers, such as DigitalOcean's Paperspace, AWS EC2, and Google Cloud Platform's Cloud GPUs, it

becomes evident that they come with significantly higher costs. For instance, the cheapest plan among these providers can be as high as 0.6 USD per hour (~4.7 HKD). In contrast, utilizing the official Mistral AI's API offers a more cost-effective solution. By dynamically adjusting compute usage based on tokens parsed and generated, Mistral AI's API ensures elasticity and reduces the operational expenses of the application.

### 5.3 Message Queue Disconnection

The Pika Python library, the official Python library for working with RabbitMQ, was used to connect to the Message Queue Server, which supports Machine Learning services. The incoming reviews were read from the message queue and processed by the deployed ML Models. However, random disconnections were experienced, without any error messages from both the Python client and the Message Queue Server. The logs revealed that the disconnection frequency varied from 10 minutes to 3 hours after redeployment. There were no error messages from both the Python client and the Message Queue Server. Logs reveal that disconnection occurs at random frequencies, ranging from 10 minutes to 3 hours after redeployment.

Many solutions for common problems were attempted to fix the issue, including:

- **Enforcing Mandatory Delivery Confirm (Acknowledgement)**: This solution aimed to prevent data loss due to disconnections by using the transactional mode of Pika. The producer would send a batch of messages and wait for the broker to confirm that they were received and persisted. The producer would block until the broker responded with a **commit_ok** message, indicating that the transaction was successful. If the connection was closed or the broker returned an error, the producer would catch the exception and retry the transaction.
- **Reverting to Single Threading**: This solution aimed to reduce the complexity and the overhead of managing multiple connections and channels by using a single-threaded approach for both the producer and the consumer. The producer and the consumer would use the *pika.BlockingConnection* class, which provides a simple and synchronous interface for interacting with the broker. The producer and the consumer would use the *channel.basic_publish()* and *channel.basic_consume()* methods to send and receive messages, respectively. The *channel.basic_consume()* method would block until a message was delivered, and invoke a callback function to process the message.

- **Mandatory Flag (Return message on Failure)**: This solution aimed to handle any messages that could not be routed to a queue by using the mandatory flag of Pika. The producer would instruct the broker to return any unrouteable messages, which could happen if the queue did not exist, or if the queue was full or had reached its limit. The producer would set the mandatory parameter to True when calling the *channel.basic_publish()* method. The producer would also register a callback function with the *channel.add_on_return_callback()* method, which would be invoked when the broker returned a message. The callback function would then handle the returned message, such as logging it, retrying it, or discarding it.

However, none of the mentioned solutions solved the disconnection issue.

The final solution that resolved the unpredictable disconnection issue was to disable the heartbeat check (See Figure 105). The heartbeat check is a mechanism that allows the broker and the client to detect and close stale connections. The broker and the client exchange heartbeat frames at regular intervals, and if either side does not receive a heartbeat frame within a specified timeout, it will close the connection. However, this mechanism can also cause problems if the network is unreliable, or the client is busy processing messages. To use this solution, the Blocking Connection for the consumer was set to not use the heartbeat check. This would tell the broker not to expect any heartbeat frames from the client. With further testing, disabling the heartbeat check did not lead to any additional packet loss or drop.

```
connection = pika.BlockingConnection(pika.ConnectionParameters(
    host=[redacted], port=[redacted], credentials=credentials, heartbeat=0))
```

*Figure 105: Code snippet to initiate the RabbitMQ connect with heartbeat check disabled.*

# 6 Limitations

This section discusses the limitations faced during the development process of the project, including Noise in Review Data (Section 6.1) and LLM Hallucination (Section 6.2).

## 6.1 Noise in Review Data

The dataset employed for training our machine learning models is predominantly derived from user-generated reviews on the Steam platform, complemented by a curated collection of professional critiques. Despite rigorous preprocessing efforts to refine the dataset by excluding reviews of insufficient length or those heavily laden with stop words, a substantial portion of the player-generated content persists. This content frequently encompasses spam or sarcastic commentary, thereby introducing a significant degree of noise into the dataset and potentially impairing the performance of the models.

To elucidate, the following are more granular explanations and instances of the types of noise encountered within the dataset:

- **Non-relevant text**: This category encompasses segments of reviews that are tangential to the actual gameplay experience. Such text may consist of personal narratives, broad observations about the gaming sector, or discussions on topics unrelated to the game under review.
- **(Word) Spam**: Certain reviews are characterized by the repetition of words or phrases that fail to impart substantial information. The dataset contains numerous instances where terms like “GOAT” (an acronym for Greatest of All Time) or “Cake” (a reference to in-game lore) are employed repetitively without adequate context or rationale, posing challenges for the model’s interpretative capabilities.
- **Language Inconsistency**: The reviews display a diverse array of linguistic styles, ranging from formal and coherent to casual and disjointed. The prevalence of internet jargon, meme references, and sentences that amalgamate multiple languages further complicates the dataset.

These manifestations of noise pose considerable challenges for both our trained Sentiment Analysis Model and the pre-trained Large Language Model. They have the potential to obscure pertinent information, leading to inaccuracies in sentiment analysis, keyword extraction, and content summarization tasks. To address these challenges, the implementation of more advanced preprocessing techniques is warranted. Such measures could include context-sensitive filtering, the deployment of sophisticated sentiment analysis tools capable of

discerning sarcasm, and algorithms adept at managing mixed-language content. Moreover, refining the selection process for professional reviews and integrating user ratings with textual feedback may offer a more balanced dataset, thereby enhancing the learning efficacy of the models. The adoption of these strategies is anticipated to mitigate the impact of dataset noise and improve the overall accuracy and reliability of the machine learning models in question.

### 6.2 LLM Hallucination

"Hallucination" is used to describe a situation where a language model generates content that does not align with the input provided. This could manifest as the creation of information or the extraction of keywords that were never mentioned in the game review or the context of the game being discussed. Even when utilizing advanced pre-trained open-source language models, such as Llama2 from Meta or Mistral from Mistral AI, this issue can arise.

These models are designed to generate content by predicting the next sequence of words based on the input they receive. To control the level of creativity and randomness in the output, a parameter known as "temperature" is adjusted. A lower temperature setting is supposed to result in more predictable and less inventive content. However, even with a reduced temperature, these language models can sometimes produce outputs that are imaginative or creative but lack factual accuracy. This is particularly challenging when dealing with game reviews that feature new genres or game mechanics that the model has not been trained on, leading to difficulties in accurately categorizing or summarizing the content.

The following shows an example from a real user review on the game "Monster Hunter: World" on Steam (See Figure 106), where the LLM reported keywords on aspects, such as Sound and Graphics & Art Design, where there is no mention of these aspects at all. In this short review, only the gameplay aspect of the game is discussed. Although the LLM correctly identified the keywords related to the gameplay aspect and the overall game experience, it struggled to deal with the hallucinations of others.

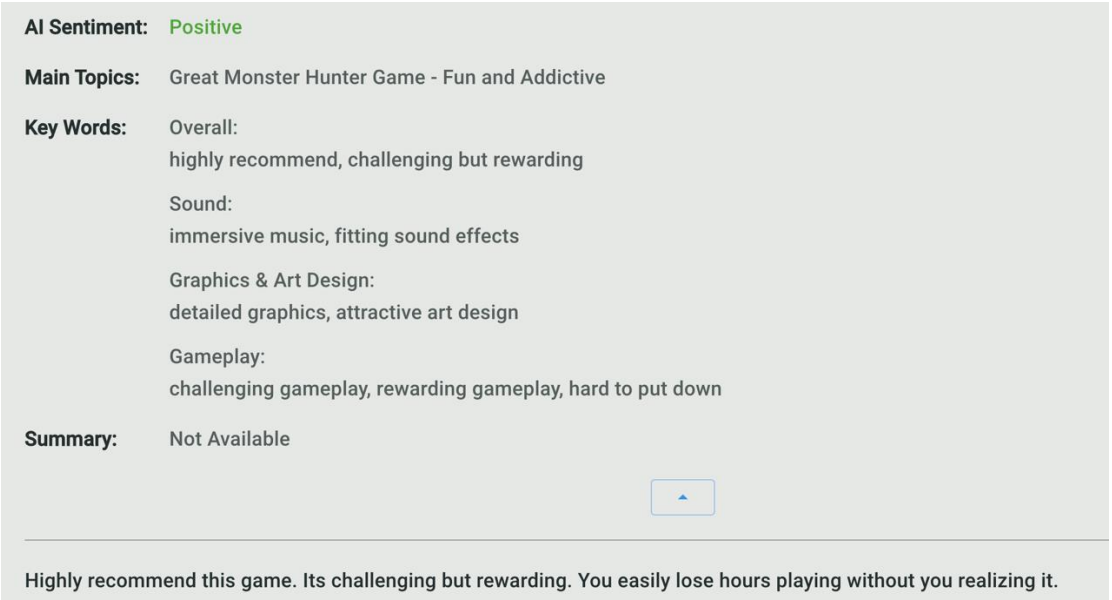



*Figure 106 Example of LLM hallucination, generating keywords that do not exist in the review*

This phenomenon is particularly apparent when the review is short or very long. Short reviews may not provide enough context or details, possibly leading the LLM to fill in gaps with assumptions that may not be accurate to what is mentioned. Conversely, very long reviews might contain too much information, some of which could be tangential or less relevant and lead to the LLM focusing on the wrong details by the verbosity, providing incorrect results.

While various prompt engineering techniques have been employed, some degree of hallucination persists. Until additional prompt engineering techniques are developed or future LLM enhance their ability to guard against hallucination, it remains a challenge when analyzing both short and long texts and required to process the review in various forms.

Furthermore, the efficacy of language models in generating high-quality content is inherently tied to their context size constraints. These models are designed with a maximum token limit, which dictates the extent of text they can process or “remember” at any given instance. Inputs that surpass this threshold may lead to the model overlooking critical information, thereby diminishing the content’s quality. Moreover, the model’s performance—its proficiency in comprehending and producing text—is pivotal in determining the quality of the output. Factors influencing performance include the diversity and scope of the training data, the structural design of the model, and the computational power allocated for processing. These elements collectively define the model’s operational boundaries and its ability to deliver outputs that meet high-quality standards.

# 7 Future Works

Based on the results achieved during the projects and previously mentioned limitations, the following work is suggested as fields of study and improvement, including Integration with Business Intelligence Tools (Section 7.1), LLM for 3-Way Sentiment Analysis (Section 7.2), Topic Modeling Spam Filtering (Section 7.3) and Game Aggregated Review Automation (Section 7.4).

## 7.1 Integration with Business Intelligence Tools

The integration of Business Intelligence (BI) tools with Spring Boot for the purpose of analyzing game reviews presents a multitude of benefits that surpass the capabilities of direct Java programming. BI tools are inherently designed to manage and interpret complex datasets, offering a robust and scalable framework suitable for the voluminous data generated by game reviews. These tools are equipped with a suite of pre-configured features, such as sentiment analysis, frequency of word occurrences, and the identification of prevalent topics, which can drastically diminish the time and resources required for development in comparison to manual coding in Java.

Furthermore, BI tools incorporate sophisticated algorithms that specialize in data mining and machine learning, facilitating a more nuanced and profound analysis of player feedback. This, in turn, can refine the caliber of recommendations provided and elevate the overall user experience within the gaming ecosystem. The interactive dashboards and visual representations that BI tools offer translate complex data into actionable insights, enabling stakeholders to make informed decisions based on the analysis of game reviews.

A pivotal advantage of BI tools is their seamless integration with existing data ecosystems and infrastructures. They can effortlessly interface with Spring Boot applications, thereby enabling instantaneous analysis and reporting. This symbiosis also promotes superior data management and security protocols, as BI tools come with built-in functionalities that regulate data access and safeguard confidential information.

Modern BI solutions, such as Tableau and PowerBI, are adept at accessing data from a variety of sources, including databases and message queues, without disrupting the operational threads of concurrent applications. These tools support the exportation of

visualizations in multiple formats, such as images and PDFs, obviating the need for front-end applications to depend on third-party libraries for visualization purposes.

Lastly, the adoption of BI tools fosters collaboration across various departments, including development, marketing, and customer service. The collective insights and analyses can synchronize organizational strategies and decisions, propelling a unified, data-centric approach to game development and marketing. The strategic use of BI tools in conjunction with Spring Boot for game review analytics can streamline data processing, deepen analytical insights, bolster decision-making processes, and enhance interdepartmental cooperation, all of which are instrumental in the advancement and success of the gaming platform.

### 7.2 LLM for 3-Way Sentiment Analysis

**LLM for Sentiment Analysis**

Leveraging Large Language Models (LLMs) for game review sentiment analysis presents a significant advantage over binary classification with BERT, particularly due to the nuanced nature of game reviews. LLMs' ability to discern neutral sentiments is crucial, as not all reviews are polarized; many express mixed or ambivalent feelings that are better captured by a neutral category. This three-way classification (Positive, Negative, Neutral) aligns more closely with the spectrum of human emotions and provides a more granular understanding of player feedback.

Our platform's sentiment analysis, as outlined in Section 4.1, relies on a self-trained BERT model that is limited to binary sentiment classification—positive or negative. This model has shown excellent performance across user reviews, which tend to be lower in quality and higher in noise, as well as professional critic reviews, which are higher in quality and diversity. However, the binary nature of our training data from the Steam platform, which only offers a binary recommendation value, restricts us to a binary output. The absence of a substantial dataset comprising neutral reviews poses a challenge to self-training a model for three-way classification.

Furthermore, LLMs enhance sentiment analysis by incorporating additional context such as the game's name, its description, and the overall user perception. This broader context allows LLMs to understand the sentiment about specific game features or aspects

mentioned in the reviews, leading to a more accurate and comprehensive analysis. By considering these additional factors, LLMs can provide insights that go beyond mere sentiment classification, offering a deeper dive into the subtleties of user opinions and experiences. This holistic approach is particularly beneficial for developers and publishers looking to glean actionable insights from player feedback.

Moreover, LLMs excel in contextually analysing text, taking into account surrounding words and sentences to accurately infer sentiment. This feature is particularly crucial in game reviews, where context can significantly shift the sentiment conveyed. For instance, the phrase "This game has an incredibly easy learning curve" might be perceived as positive, negative, or neutral, contingent upon the context set by the game's genre or the audience's expectations. Additionally, LLMs' proficiency in processing and generating human-like text equips them to adeptly handle idiomatic expressions, sarcasm, and other subtle forms of language prevalent in game reviews. This capability ensures a more precise reflection of the reviewers' genuine opinions and sentiments toward the game, providing invaluable insights for developers and marketers aiming to comprehend their audience.

**LLM as supporting tool for external Sentiment Analysis Model**

LLMs' adaptability to new genres or keywords surpasses that of our self-trained BERT model, which is constrained by its finite corpus. LLMs, with their expansive knowledge base, can seamlessly adjust to the ever-changing lexicon of the gaming world. They can understand and apply novel terms and concepts that arise, ensuring that sentiment analysis remains relevant and accurate, even as the gaming industry continues to innovate and expand. This adaptability makes LLMs an indispensable tool for capturing the full spectrum of player feedback, catering to the nuanced and evolving nature of game reviews.

In the realm of sentiment analysis for game reviews, the deployment of Large Language Models (LLMs) marks a significant advancement. LLMs can be harnessed to meticulously analyse extensive collections of game reviews, extracting the subtle nuances within the text to assign accurate sentiment labels—Positive, Negative, or Neutral. This process, known as Sentiment Generation, leverages the LLMs' deep understanding of context and language subtleties, enabling them to capture the full spectrum of emotions expressed in player feedback. Once the sentiment labels are generated, the next step is Dataset Creation. This involves compiling the original review texts and their corresponding sentiment labels into

a new dataset. This dataset, enriched with the LLMs' sentiment annotations, serves as a foundational resource for further analysis and model training.

The enriched dataset paves the way for Model Training. Unlike the previous binary classification models, the new dataset facilitates the training of a more sophisticated sentiment analysis model—potentially a BERT model or another advanced machine learning algorithm—that can classify sentiments into three distinct categories. This tripartite classification system is more aligned with the complex emotional responses found in game reviews.

To ensure the model remains relevant and up-to-date, Continuous Learning is essential. As LLMs persist in evaluating new reviews and updating sentiment labels, the dataset can be regularly refreshed with this new information. Consequently, the sentiment analysis model can undergo periodic retraining with the latest data, reflecting the evolving trends and linguistic nuances of the gaming community. The use of an LLM-generated dataset for training a separate model introduces an element of Self-Control. This approach allows for a greater degree of customization and fine-tuning of the training process, model parameters, and sentiment classification standards to meet specific requirements or objectives. It affords the platform autonomy over the sentiment analysis, tailoring it to its unique needs.

Lastly, the Feedback Loop is a critical component of this system. Monitoring the performance of the self-trained model is crucial, and any observed inaccuracies or misclassifications in sentiment analysis can be channelled back to the LLM. This feedback mechanism facilitates the continuous refinement and enhancement of the LLM's sentiment generation capabilities, ensuring a consistently high level of accuracy and reliability in sentiment analysis. This iterative process not only improves the model's performance but also contributes to the LLM's learning and adaptation over time.

### 7.3 Topic Modeling Spam Filtering

Our current implementation of Spam Detection solely relies on a generic prompt, as described (see Section 4.2), that queries whether a review is spam, without delving into the specifics of the content or its relevance to aspects of the game. In addition, reviews can be flagged as not a Spam despite it not being related to the game at all. This rudimentary

approach may not be sufficient for the nuanced task of filtering out reviews that lack context or fail to address specific game aspects.

This Spam Detection system can be enhanced, by adapting LLM in the detection process. LLM can be used to provide a more sophisticated analysis by examining the content of reviews for mentions of specific game aspects, such as gameplay mechanics, graphics quality, story depth, and overall user experience. By doing so, LLMs can effectively filter out reviews that do not contribute meaningful insights or constructive criticism, which are often characteristics of spam. For instance, an LLM-enhanced system could analyze a review to determine if it discusses the game's controls or soundtrack, rather than simply labeling it as spam based on superficial criteria. This would ensure that the reviews retained for sentiment analysis and further consideration are those that genuinely reflect players' opinions and experiences, providing valuable feedback to game developers and the community.

Incorporating LLMs into the spam detection process would thus allow for a more refined and context-aware approach. It would enable the system to discern the relevance and quality of the content within reviews, ensuring that only those that provide a substantive discussion of the game are included in the analysis. This adaptation would represent a significant improvement over the current basic prompt system, leading to more accurate and useful sentiment analysis outcomes.

### 7.4 Game Aggregated Review Automation

In Section 4.3, the limitations of our current system were discussed where games with a substantial number of user reviews and existing critic (professional) reviews were allowed to have an "Aggregated Review." These summaries consolidate all stored reviews into a concise format for easy digestion.

To augment the efficacy of our system, forthcoming endeavors should prioritize the development of an automated mechanism for generating aggregated reviews. Establishing an equilibrium between consumer feedback and professional critiques is pivotal for maintaining the integrity and precision of these summaries. To this end, we propose a multifaceted approach:

**Scraping Critic Reviews**: To amass a robust corpus of professional critiques, an automated scraping system is essential. This system would not only collect reviews but also implement filters to ensure the curation of high-quality content.

**Topic Extraction**: The extraction of prevalent themes from reviews is a critical step. Currently reliant on manual oversight, the adoption of a sophisticated Language Model (LLM) could significantly expedite and refine this process.

**Automation Strategies**:

**Spring Batch**: The incorporation of Spring Batch would facilitate the automated batch processing of reviews, enhancing efficiency

**Cron Jobs**: the utilization of cron jobs for the scheduled execution of the aggregation function would ensure consistent updates at predetermined intervals.

**Message Queue Integration**: By repurposing the existing "send to message queue" functionality, Machine Learning models can be activated and triggered to process reviews and the generation of comprehensive summaries.

A delicate balance must be struck between the performance overhead associated with regularly refreshing the aggregated reviews and the necessity of their timeliness—a challenge that remained unresolved during the developmental phase. Consequently, the initiation of game aggregated reviews was conducted manually for demonstration and evaluation. A version primed for production should aspire to routinely update the aggregated reviews without imposing a significant burden on the backend application or incurring excessive costs associated with LLM operations.

In essence, the automation of game review aggregation promises to substantially bolster our system's utility, delivering distilled and informative aggregated reviews that cater to the informational needs of users. This advancement is anticipated to furnish both developers and gamers with a more nuanced understanding of the collective sentiment surrounding a game, thereby enriching the gaming community's discourse and decision-making processes.

## 8 Schedule

The project was completed following the planned schedule (See Table 7).

| Period | Work to be done |
|---|---|
| Sep | - Define Requirements for Web App and ML Models<br>- Literature Review on NLP and ML Architectures |
| Oct | - Designed Web App & DB Structure<br>- Performed Data Scraping for Model Training<br>- Performed Software Architecture Design<br>- Designed Use-Case Diagram |
| Nov – Dec | - Implemented Sign Up & Sign In Page with JWT<br>- Implemented Add Games / Reviews Page<br>- Trained & Evaluated Sentiment Analysis Model<br>- Designed Landing Page, Game Page, and Analytic Page |
| Jan – Feb | - Integrated the Sentiment Analysis model into our application.<br>- Implemented Landing Page and Game Dashboard Page<br>- Implemented Topic Modelling Model<br>Implemented Keyword Extraction model |
| Feb – Mar | - Fine-tuned Topic Modelling Model<br>Fine-tuned Keyword Extraction model |
| Mar – Apr | - Fine-tuned all models and their integration.<br>- Debug and Refactor Code<br>- Prepared for the Final Presentation<br>- Prepared Demo and Demo Video |

*Table 7 Proposed Schedule for the project*

## 9 Work Distribution

The contribution of each member in the project is listed (See Table 8).

| Student | Contributions |
|---|---|
| Lee Chi Ho | - Backend Architecture Design<br>- Cloud Solution Design and Setup<br>- Database Design and Implementation<br>- Spring Boot Server Implementation<br>- DevOps Design |
| Cheng Pak Yim | - Research, Implement, and Evaluate all Sentiment Analysis models.<br>- Research Topic Modeling models<br>- Research Keyword Extraction models<br>- Implement Python client side of the NLP message queue. |
| Siu Yuk Shing | - Web Application Design<br>- Frontend Implementation<br>- Frontend Libraries and Packages Setup |

*Table 8 Work Distribution Table of the project*

## 10 Conclusion

The project at hand represents a significant leap in the application of Natural Language Processing (NLP) within the gaming industry, particularly in the realm of game review analytics. Historically, the utilization of NLP in this sector has been minimal; however, this initiative has successfully developed a web application that leverages NLP to provide developers with automated tools for the analysis and aggregation of game reviews. This integration of NLP into a scalable system has proven to accelerate the review analysis process significantly.

The development journey culminated in the creation of a comprehensive web application, meticulously engineered to include authentication, game reviews, discussion features that automates the analysis process and provide detailed analytics with different forms of visualizations. The backend infrastructure utilizes a robust CI/CD pipeline on cloud platforms, ensuring seamless updates from development to production, thus enhancing the development process and speeding up deployment of the backend solutions. The empirical data gathered throughout the project's duration highlight the precision and efficiency of NLP techniques such as Sentiment Analysis, Topic Modeling, Keyword Extraction, and Idea Summarization and could achieve high degree of accuracy, comparable to manual-processing. These results have demonstrated their ability to deliver high-quality analytical insights, significantly reducing the manual effort required to filter out spam reviews and extract meaningful causality from the data. While analysis created using different Machine Learning models could be useful for individual analytics, we also acknowledge the importance of data aggregation in understanding the overall trend of the underlying dataset. To this end, we provide a detailed analytic page that focuses on data graphing and charting to improve the readability and ease of understanding of the vast amount of data presented.

The outcomes of this project validate the premise that a well-designed system architecture can enable the seamless and cost-effective integration of NLP tools into pre-existing technological frameworks and architectures. This integration enhances the analytical prowess of game developers and deepens the understanding of game content for end-users. The project stands as a testament to the transformative potential of NLP in enriching the gaming experience through intelligent data analysis. Advancements in NLP and the introduction of new Language Learning Models (LLMs) like Meta AI’s Llama3 and Anthropic’s Claude will enhance content generation and analysis’s efficiency and accuracy. These advancements will allow for more

precise and accurate output while necessitating minimal modifications to the existing codebase for migration. This project, therefore, not only demonstrates the current capabilities of NLP in the gaming industry but also highlights the potential for future developments in this exciting field.

## References


Abdelrazek, A., Eid, Y., Gawish, E., Medhat, W., & Hassan, A. (2023). Topic modeling algorithms and applications: A survey. *Information Systems*, *112*, 102131. https://doi.org/10.1016/j.is.2022.102131

Al Mursyidy Fadhlurrahman, J., Herawati, N. A., Widya Aulya, H. R., Puspasari, I., & Utama, N. P. (2023, 2023/10/10/). Sentiment Analysis of Game Reviews on STEAM using BERT, BiLSTM, and CRF. 2023 International Conference on Electrical Engineering and Informatics (ICEEI),

Bianchi, F., Terragni, S., & Hovy, D. (2021, 2021/08//). Pre-training is a Hot Topic: Contextualized Document Embeddings Improve Topic Coherence. ACL-IJCNLP 2021,

Birjali, M., Kasri, M., & Beni-Hssane, A. (2021). A comprehensive survey on sentiment analysis: Approaches, challenges and trends. *Knowledge-Based Systems*, *226*, 107134. https://doi.org/https://doi.org/10.1016/j.knosys.2021.107134

Churchill, R., & Singh, L. (2022). The Evolution of Topic Modeling. *ACM Comput. Surv.*, *54*(10s), Article 215. https://doi.org/10.1145/3507900

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: arXiv.

Egger, R., & Yu, J. (2022). A Topic Modeling Comparison Between LDA, NMF, Top2Vec, and BERTopic to Demystify Twitter Posts [Methods]. *Frontiers in Sociology*, *7*. https://doi.org/10.3389/fsoc.2022.886498

Fachrul, K., Yuliana, R., Laila, Z., & Hammad, J. (2022). Comparing LSTM and CNN Methods in Case Study on Public Discussion about Covid-19 in Twitter. *International Journal of Advanced Computer Science and Applications*. https://doi.org/10.14569/ijacsa.2022.0131048

Gan, L., Yang, T., Huang, Y., Yang, B., Luo, Y. Y., Richard, L. W. C., & Guo, D. (2024, 2024//). Experimental Comparison of Three Topic Modeling Methods with LDA, Top2Vec and BERTopic. Artificial Intelligence and Robotics, Singapore.

Grootendorst, M. (2022). BERTopic: Neural topic modeling with a class-based TF-IDF procedure. In: arXiv.

Guzsvinecz, T., & Szűcs, J. (2023). Length and sentiment analysis of reviews about top-level video game genres on the steam platform. *Computers in Human Behavior*, *149*, 107955. https://doi.org/https://doi.org/10.1016/j.chb.2023.107955

Jaiswal, P., Setia, H., Raghuwanshi, P., & Randhawa, P. (2023). A Natural Language Processing Model for Predicting Five-Star Ratings of Video Games on Short-Text Reviews. *Engineering Proceedings*, *59*(1), 58. https://www.mdpi.com/2673-4591/59/1/58

Khan, M. Q., Shahid, A., Uddin, M. I., Roman, M., Alharbi, A., Alosaimi, W., Almalki, J., & Alshahrani, S. M. (2022). Impact analysis of keyword extraction using contextual word embedding. *PeerJ Computer Science*, *8*, e967. https://doi.org/10.7717/peerj-cs.967

Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. In: arXiv.

Li, X., Li, X., Rodolfo, C. R., & Shi, X. (2022). GloVe-CNN-BiLSTM Model for Sentiment Analysis on Text Reviews. *Journal of Sensors*. https://doi.org/10.1155/2022/7212366

Lin, D., Bezemer, C.-P., Zou, Y., & Hassan, A. E. (2019a). An empirical study of game reviews on the Steam platform. *Empirical software engineering : an international journal*, *24*(1), 170-207. https://doi.org/10.1007/s10664-018-9627-4

Lin, D., Bezemer, C.-P., Zou, Y., & Hassan, A. E. (2019b). An empirical study of game reviews on the Steam platform. *Empirical Software Engineering*, *24*(1), 170-207. https://doi.org/10.1007/s10664-018-9627-4

Maisa, J. A.-K., Marwah, A., Mariam, M. B., & Bayan, A.-H. (2023). Sentiment Analysis for People's Opinions about COVID-19 Using LSTM and CNN Models. *Int. J. Online Biomed. Eng.* https://doi.org/10.3991/ijoe.v19i01.35645

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. In: arXiv.

Morgenthaler, S. (2009). Exploratory data analysis. *WIREs Computational Statistics*, *1*(1), 33-44. https://doi.org/https://doi.org/10.1002/wics.2

Öğüt, M. S. S. (2021, 17-19 Nov. 2021). A Performance Study Depending on Execution Times of Various Frameworks in Machine Learning Inference. 2021 15th Turkish National Software Engineering Symposium (UYMS),

Prusa, J., Khoshgoftaar, T. M., & Seliya, N. (2015, 9-11 Dec. 2015). The Effect of Dataset Size on Training Tweet Sentiment Classifiers. 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA),

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving Language Understanding by Generative Pre-Training.

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In: arXiv.

Röder, M., Both, A., & Hinneburg, A. (2015). Exploring the Space of Topic Coherence Measures. *WSDM 2015 - Proceedings of the 8th ACM International Conference on Web Search and Data Mining*, 399-408. https://doi.org/10.1145/2684822.2685324

Rogers, A., Kovaleva, O., & Rumshisky, A. (2021). A Primer in BERTology: What We Know About How BERT Works. *Transactions of the Association for Computational Linguistics*, *8*, 842-866. https://doi.org/10.1162/tacl_a_00349

Ruseti, S., Sirbu, M.-D., Calin, M. A., Dascalu, M., Trausan-Matu, S., & Militaru, G. (2020, 2020). Comprehensive Exploration of Game Reviews Extraction and Opinion Mining Using NLP Techniques.*Advances in Intelligent Systems and Computing*

Sobkowicz, A. (2017). *Steam Review Dataset (2017)*. https://doi.org/https://zenodo.org/doi/10.5281/zenodo.1000884

Srivastava, A., & Sutton, C. (2017). Autoencoding Variational Inference For Topic Models. In: arXiv.

Stepien, K. (2021). *Topic Modelling and Data Analysis: Impact of using topic modelling on game reviews* https://eprints.lincoln.ac.uk/id/eprint/49978/1/Thesis_-_Konrad_Stepien_Corrections_16-01-2022%20(1).pdf

Sun, C., Qiu, X., Xu, Y., & Huang, X. (2019, 2019). How to Fine-Tune BERT for Text Classification?*Lecture Notes in Computer Science*

Tan, J. Y., Chow Sai Kit, A., & Tan, C. W. (2021). *Sentiment Analysis on Game Reviews: A Comparative Study of Machine Learning Approaches*.

Valve. (2023). *Steamworks API Reference* https://partner.steamgames.com/doc/webapi

Viggiato, M., Lin, D., Hindle, A., & Bezemer, C.-P. (2022). What Causes Wrong Sentiment Classifications of Game Reviews? *IEEE Transactions on Games*, *14*(3), 350-363. https://doi.org/10.1109/TG.2021.3072545

Wenxuan, Z., & Yuxuan, W. (2022). Semantic sentiment analysis based on a combination of CNN and LSTM model. *2022 International Conference on Machine Learning and Knowledge Engineering (MLKE)*. https://doi.org/10.1109/mlke55170.2022.00041

Yu, Y., Nguyen, B. H., Dinh, D. T., Yu, F., Fujinami, T., & Huynh, V. N. (2022, 2022). A Topic Modeling Approach for Exploring Attraction of Dark Souls Series Reviews on Steam. AHFE (2022) International Conference,

Zhang, Y., & Wallace, B. (2016). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. In: arXiv.