The Hashemite University
Faculty of Prince Al-Hussein Bin Abdallah II For Information Technology
Information Technology Department

Steam Recommendation System and Sentiment Analysis

A project submitted
in partial fulfillment of the requirements for the
B.Sc. Degree in Data science and Artificial Intelligence

By

Wael Walid Siam (1932018)
Hatem Yousef Abu Sadeh (2132159)
Jamal Yaser Al Elaumi (2132132)
Ghufran Mahmoud Abu Zaid (2139724)

Supervised by

Dr. Zaher Ibrahim Saleh

May/2024

CERTIFICATE

It is hereby certified that the project titled < ***Recommendation System and Sentiment Analysis*** >, submitted by undersigned, in partial fulfillment of the award of the degree of “Bachelor in Data science and Artificial Intelligence” embodies original work done by them under my supervision.

All the analysis, design and system development have been accomplished by the undersigned. Moreover, this project has not been submitted to any other college or university.

Student name (Student number)	**Signature**
Wael (1932018)	
Hatem (2132159)	
Jamal (2132132)	
Ghufran (2139724)	

ABSTRACT

In this project, we created a platform that combines a game recommendation system, sentiment analysis, and an AI chatbot to enhance the experience for both players and developers on Steam. Steam, launched by Valve Corporation in 2003, originally started as a platform for updating Valve's games and expanded to include games from other companies in 2005. Today, it is the largest PC gaming platform, with over 34,000 games and 132 million monthly users (as of 2021, according to SteamSpy).

With so many games available, users often struggle to find games they'll enjoy. Meanwhile, developers can find it challenging to keep up with feedback, as some games receive millions of reviews. Our platform addresses these issues through a recommendation system that helps users discover games based on their preferences and sentiment analysis tools that assist developers in understanding player feedback.

Additionally, we implemented a chatbot that simplifies the platform experience. The chatbot uses AI to interact with users, offering personalized game recommendations based on their descriptions and preferences. By leveraging the recommendation system, it helps users find games they are likely to enjoy, making it easier for players to discover new titles and enhancing their overall gaming experience.

Figure 1. *Steam Logo*

ACKNOWLEDGEMENTS

Very special thanks to our supervisor for the great guidance and assistance, which were crucial in completing this project.

We would like to thank our committee members who were generous with their expertise and precious time.

Last but not least, our deep gratitude to our families who have always supported us.

TABLE OF CONTENTS

ABBREVIATIONS

- **Artificial Intelligence (AI):** A set of technologies that enable computers to perform a variety of advanced functions, including the ability to see, understand and translate spoken and written language, analyze data, make recommendations, and more.
- **Application Programming Interface (API):** A set of rules or protocols that enables software applications to communicate with each other to exchange data, features, and functionality.
- **Natural Language Processing (NLP):** A machine-learning technology that gives computers the ability to interpret, manipulate, and comprehend human language.
- **TF-IDF:** Short for the Term Frequency–Inverse Document Frequency, is a measure of the importance of a word to a document in a collection or corpus, adjusted for the fact that some words appear more frequently in general.
- **Term Frequency (TF):** The term frequency $\mathrm{tf}_{t,d}$ of term t in document d is defined as the number of times that t occurs in d.
- **Secure Socket Layer (SSL):** A Security protocol that provides privacy, authentication, and integrity to internet communications.
- **Natural Language Toolkit (NLTK):** A suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English written in the Python programming language. It supports classification, tokenization, stemming, tagging, parsing, and semantic reasoning functionalities.
- **Uniform Resource Locator (URL):** A reference to a resource that specifies its location on a computer network and a mechanism for retrieving it.
- **Comma-Separated Values (CSV):** A text file format that uses commas to separate values, and newlines to separate records.

- **Retrieval-Augmented Generation (RAG):** A technique for enhancing the accuracy and reliability of generative AI models with facts fetched from external sources.

DEFINITIONS

- **Steam:** A video game digital distribution service and storefront developed by Valve Corporation.
- **Steamworks:** A freely available application programming interface (API) released in 2008, and used by developers to integrate Steam's functions, including digital rights management (DRM) into their game products.
- **SteamSpy:** A Steam stats service based on Web API provided by Valve and it originated as an idea proposed by Kyle Orland from Ars Technica. SteamSpy automatically gathers data from Steam user profiles, analyzes it, and presents it in a simple, yet beautiful manner.
- **Cosine similarity:** Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. It is often used to measure document similarity in text analysis.
- **Word cloud:** An image that shows all the important words used in a text in sizes related to the frequency with which they are used.
- **Sentiment analysis:** (Also known as opinion mining or emotion AI) is the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information.
- **Recommendation system:** A class of machine learning that uses data to help predict, narrow down, and find what people are looking for among an exponentially growing number of options.
- **Text summarization:** The process of using deep learning and machine learning models to synthesize large bodies of texts into their most important parts.

- **Length normalization**: A vector can be (length-normalized) by dividing each of its components by its length.
- **Lemmatization:** A text normalization process in natural language processing (NLP) that reduces words to their base or root form.
- **Case folding:** Reduce all letters to lowercase.
- **Stop words:** They have little semantic content: *the, a, and, to, be.*
- **Tokenization:** The process of splitting text into individual units called tokens. These tokens can be (words, phrases, symbols) or other meaningful elements.
- **Sentiment polarity**: Refers to the classification of the sentiment expressed in a piece of text as positive, negative, or neutral.
- **Bilinear interpolation**: A method of image resampling that calculates the pixel value at a given point in the output image by using a weighted average of the four nearest pixels in the input image. This results in a smoother appearance compared to nearest-neighbor interpolation, which simply takes the value of the closest pixel.
- **TextBlob**: A Python library for processing textual data. It provides a simple API for common natural language processing (NLP) tasks, such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more. TextBlob is built on top of the Natural Language Toolkit (NLTK) and the Pattern library, making it a user-friendly tool for NLP tasks.
- **Sentiment Analysis by Topic:** Refers to the process of evaluating and determining the sentiment (positive, negative, or neutral) expressed in the text specifically about different topics or aspects within the text.

- **Similarity scores**: Are quantitative measures used to determine how alike two objects are. These scores are typically used in various fields such as information retrieval, recommendation systems, machine learning, and natural language processing. The similarity can be computed between different types of data such as text, images, vectors, or more complex structures.
- **Streamlit:** An open-source Python library that makes it easy to create and share beautiful, custom web apps for machine learning and data science.

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

1.1 Project Overview

The gaming industry has grown rapidly, reaching a value of over $197 billion in 2022, up from $155 billion in 2020, with around 3.2 billion gamers worldwide. Among popular platforms like Epic Games, PlayStation, and Xbox, Steam stands out as the leading platform for PC gaming.

In this project, we built a system to recommend games and analyze reviews. The recommendation feature uses user preferences to suggest suitable games, while Natural Language Processing (NLP) summarizes player feedback, highlighting strengths and areas for improvement.

Additionally, we developed a chatbot that helps users find games by recommending titles based on their descriptions, making it easier to discover games aligned with their interests.

1.2 Project Motivation

1) The platform helps AAA and indie game developers manage community feedback more efficiently, using sentiment analysis and an AI-powered chatbot to engage with players and gain insights from reviews.

2) For users, the platform reduces decision fatigue and saves time by providing personalized game recommendations through the chatbot, making it easier to find games similar to their favorites.

1.3 Problem Statement

As the number of available games on platforms like Steam continues to grow, users often encounter difficulty in navigating through the millions of games to discover titles that align with their preferences. As we know, in the process of game development, user reviews play a pivotal role in guiding purchase decisions and shaping perceptions of game quality. However, among all the reviews you see on

platforms like Steam, negative feedback can often overshadow positive feedback, posing a challenge for both developers and consumers. Effectively categorizing and understanding the underlying themes and issues within negative reviews can provide valuable insights for developers to address shortcomings and for consumers to make informed decisions.

1.4 Project Aim and Objectives

A platform that eases the way for gamers to customize their games recommendations in a user-friendly platform leveraging data and content-based filtering techniques, also, aids in making sure that their voices are heard by the development teams by making it easier for them to have an abstract on the reviews by the sentiment by topic.

Objectives:

1. Develop a Recommendation Engine.
2. Integrate Natural Language Processing (NLP).
3. Create a Chatbot Interface.
4. Perform Sentiment Analysis by Topic.
5. Visualize Common Words in Reviews.
6. Provide Developer Tools to Get Insights.

1.5 Project Expected Output

We Want to deliver a complete game recommendation and review analysis system that contains the following features:

- **Personalized Game Recommendations:** Highly accurate recommendation system that suggests games based on individual user preferences.

- **Interactive Chatbot:** A chatbot that allows users to describe their favorite games and get a personalized recommendation based on the

description provided, and for the developers to receive insight about their game reviews.

- **Precise Sentiment Analysis:** Sentiment analysis of game reviews, categorized by particular aspects like gameplay, story, and performance.

- **Review Visualization Tools:** Word cloud and other visualizations that show common aspects and sentiments in the reviews.

- **Developer Insights:** Helping developers gain insight from the reviews, allowing them to improve their games based on the feedback.

1.6 Project Schedule

Table 1. *Project Schedule*

Subject	Date
Presented the project idea to our supervisor, and it was approved.	17/3/2024
Ideas similar to ours were observed, and inspiration was drawn from certain concepts and libraries, as well as their shortcomings, which were addressed in the development of this project.	20/3/2024
Process started by organizing ideas and searching for sources to obtain data.	25/3/2024
Completed the recommendation system.	2/4/2024
While working on sentiment analysis, we came up with a nice idea to display the most frequent words, so we kept the idea in mind.	15/4/2024
Began preliminary planning for the site's appearance, and it was agreed upon among the members.	15/5/2024
Took notes from our Discussants in order to improve our platform	22/5/2024
Completed the improvements in the recommendation system	22/7/2024
Started searching for sources to know how to integrate a chat-bot with our data	1/9/2024
Completed the chat-bot	15/9/2024
Completed the interface	29/11/2024

1.7 Report Organization

The rest of the report is organized as follows:

- **Chapter 2**: A review of related literature will be presented.
- **Chapter 3**: The requirements engineering and analysis will be discussed, providing insight into data collection and preprocessing.
- **Chapter 4**: We will detail the model development and architectural design.
- **Chapter 5**: The model/system evaluation and testing plan will be outlined.
- **Chapter 6**: The model deployment and integration will be addressed.
- **Chapter 7**: Finally, we will conclude the report and explore potential future work.

CHAPTER 2: LITERATURE REVIEW

In this chapter, we will talk about similar existing systems and what makes them different from our system.

2.1 Existing Systems

Tons of recommendation systems exist now, but we will talk about two of them.

1. **Steam Recommendation system:** Steam is a popular platform for video games, that helps users find new games through its recommendation system. It does that by:

    - **Reading Your History:** It checks out the games you've played and liked before. (As shown below in *figure 2*).

    - **Game Details:** It focuses on different aspects of the games, like genre, developer, and user-added tags.

    - **Similar Users:** It also looks at what other users with similar tastes are playing.

2. **Steamlytics:** It is a website that provides detailed data about games on Steam. It focuses on:

    - **Review Analysis:** Find the most common words in user reviews to get a general look at what people think.

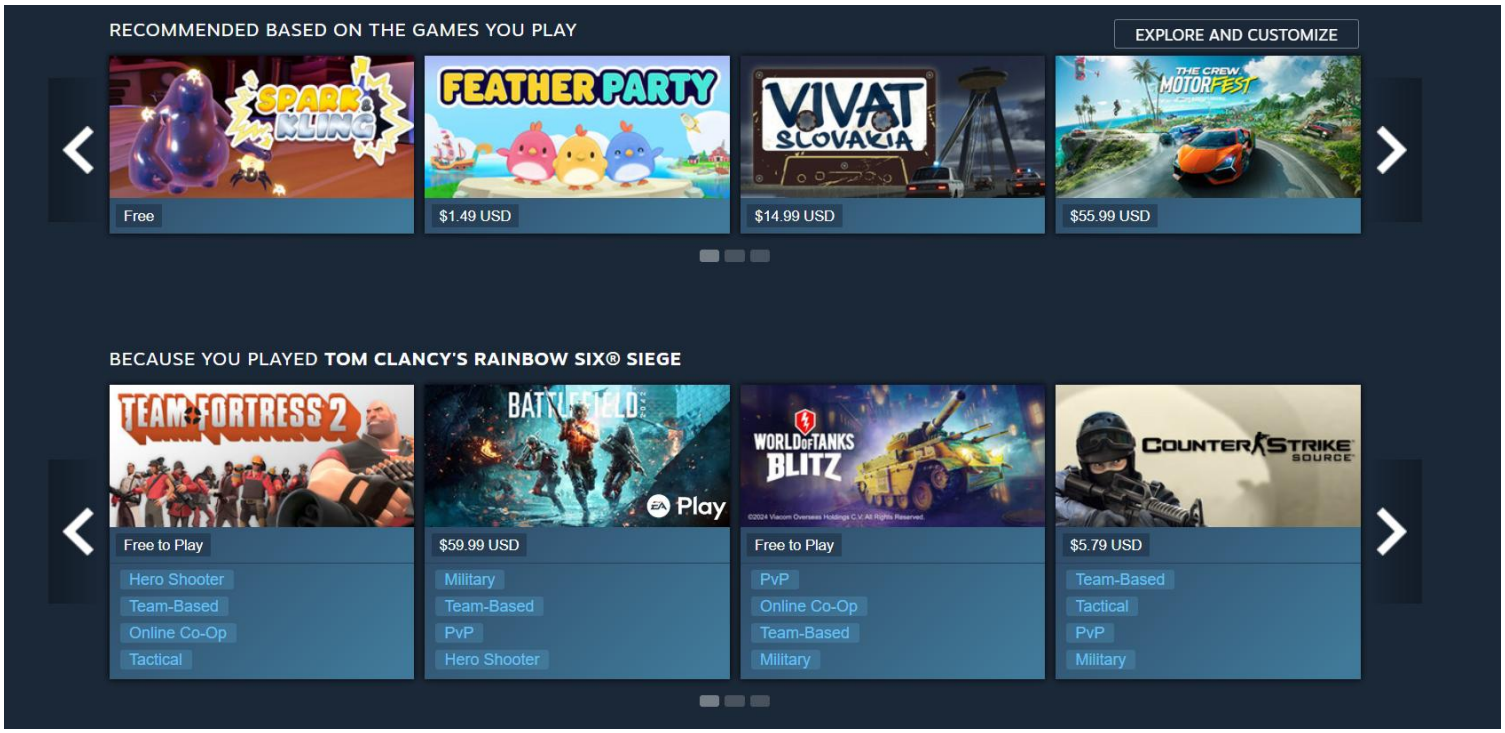

Figure 2. *Steam Recommended Games Based on Games the User Played*

2.2 Limitations of Existing Systems

The existing systems have some limitations as follows:

1. **Steam Recommendation system:**

   - **Overly General:** The recommendations can sometimes be too general and not specific enough to individual preferences.

   - **General Usage of Reviews:** It uses user reviews, but doesn't analyze the detailed feedback deeply. (As shown below in *figure 3*).

   Lack of Chatbot: The system does not include a chatbot to assist users in receiving personalized game recommendations based on their descriptions and preferences.

2. **Steamlytics:**

   - **No Personal Recommendations:** It does not offer personalized game recommendations.

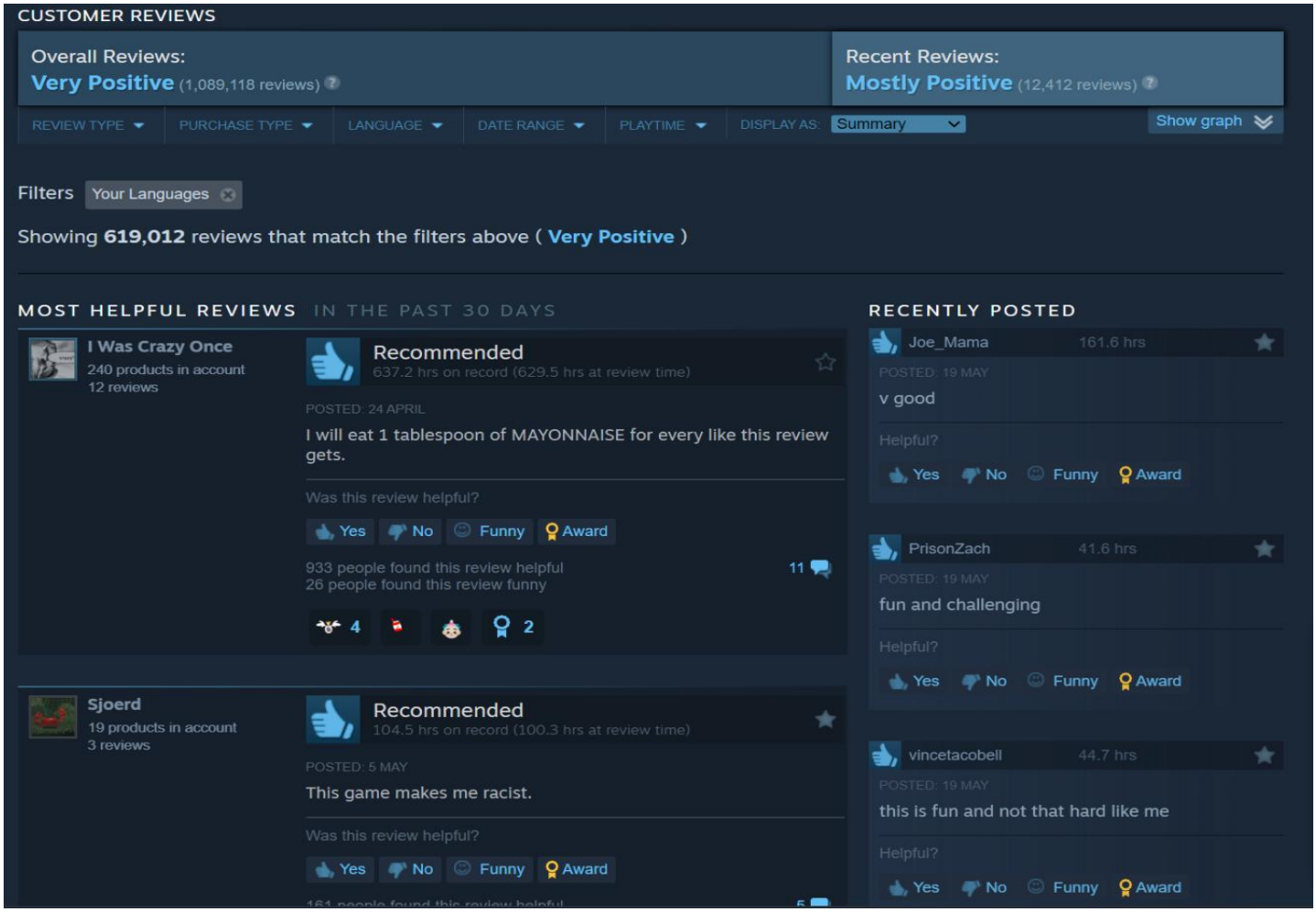

Figure 3*. A Steam Game Reviews Which Reflect Very Positive Feedback from Users*

2.3 Overall Solution Approach

Our project aims to improve these systems by offering more personalized game recommendations and deeper review analysis by providing the following services:

1. **Smarter Game Recommendations**

    - **Technique:** We use the content-based technique that provides more accurate and personalized suggestions.

    - **User Descriptions:** The users describe the type of game they want to play, and the system understands and suggests games that match their description, using Natural Language Processing (NLP).

- **Similar Game Suggestions:** If the user has a specific game in mind, the system recommends similar games by looking at a wide range of factors.

2. **In-Depth Review Analysis**

- **Word Cloud:** We Created a visual word cloud from reviews to show the most common themes and sentiments immediately.

- **Sentiment by Topic:** Instead of only counting words, we analyzed the sentiment (positive or negative) around specific topics like gameplay, story, and performance.

- **Developer Tools:** Developers can use our system to get detailed insights from user reviews, helping them understand where their game excels or needs improvements.

3. **Interactive Chatbot:**

- **For Gamers:** Users can chat with our bot to describe their ideal game and get personalized recommendations.

- **For Developers:** Developers can ask the bot for specific insights into their game's reviews, enabling targeted improvements based on comprehensive feedback analysis.

- **Key Differentiators**

    - **User-Centered Design:** Our system is designed to meet the needs of both gamers and developers, offering useful features for each of them.

    - **Advanced Analysis:** We use advanced NLP to provide deeper insights into what people are saying in reviews, and not just the most common words.

    - **Engaging and Personalized:** Our chatbot makes the experience interactive and tailored to each user's preferences, making it easier to find games the user will love.

In summary, while existing systems like Steam's recommendation engine and Steamlytics provide useful tools and insights, our project takes it a step further by integrating advanced NLP for sentiment analysis, offering more personalized recommendations, and providing interactive features that benefit both gamers and developers. This comprehensive approach ensures you get the best game suggestions and meaningful insights from user reviews.

CHAPTER 3: REQUIREMENT ENGINEERING AND ANALYSIS

This chapter focuses on preparing the groundwork for our project. We start by understanding and preparing the data and exploring various collection methods, preprocessing techniques, and integration strategies. Additionally, we explore the use of feature engineering and selection to identify relevant patterns and relationships in the data. Throughout the process, we emphasize ethical considerations and data privacy to ensure responsible project development. This chapter sets the stage for subsequent phases by establishing a solid foundation for our project's success.

3.1 Data Understanding and Preparation

3.1.1 Data Collection Methods and Sources

There are plenty of ways to get data from Steam, one of these ways is the API provided in Steamworks [1] by Valve company. Typically, an API is a great way for developers to allow access to databases and information on a server. Another great API that we used is SteamSpy API [2][3]. We gathered data from both APIs, so we needed to perform some cleaning, transformation, and integration techniques. We used Nik Davis's blog [4] which provided a huge help in gathering data about the games, we also utilized Andrew Muller's blog [5] which provided us with the knowledge needed on how to scrape reviews from games available on Steam using Python libraries like request and BeautifulSoup.

For starters, we defined a general function that handles the get requests from an API that takes two parameters; a URL parameter and a dictionary of parameters which is then passed into the get request, depending on the API requirements. We handled a couple of scenarios that helped us with the automation process, for example; we encountered an SSL (Secure Sockets Layer) Error that was solved easily by waiting a few seconds and then trying again. Another error we encountered was that we did not receive a response or the responses we received were "none". These errors occurred when too many requests were made within a short period. We were able to avoid these errors by pausing briefly between requests, which in return

guaranteed that the function returned the desired response written in JSON format, which made the processing phase easier.

We started by gathering game data to build a recommendation system, which contains a unique app ID for every app in the Steam store, although some apps might have the same name, they could never have the same app ID. This will be very helpful in identifying apps and will eventually help us merge our data tables. First, we need to generate a list of app IDs that will help us build our dataset, for that we use SteamSpy API, which should look like the following figure (Figure 4).

	appid	name
0	10	Counter-Strike
1	20	Team Fortress Classic
2	30	Day of Defeat
3	40	Deathmatch Classic
4	50	Half-Life: Opposing Force

Figure 4. *Games ID and Names*

According to SteamSpy API documentation [6], the API returns 1,000 entries per page, so we need to request multiple pages and combine the results. Then, we need to iterate over the pages and combine all the data we got into a data frame, after that, we should stop when we reach the last page. When the response contains less than 1,000 rows, that means that the last page was reached.

Now that we have the app_list data frame, we can iterate over each app's ID and request individual app data from the servers.

The idea of downloading and retrieving all game data at once would take a long time to accomplish, it would also be dangerous to attempt, as any errors or connection time-outs could cause the loss of all data. For this reason, we defined a function that would allow us to download and process the requests in batches, appending each batch to an external file and keeping track of the highest index written in a separate file. This does not only provide security by allowing us to easily restart the process if

an error is encountered, but it also means that we can complete the download across multiple sessions, as you can see in the following snippet of code.

```
def get_app_data(start, stop, parser, pause):
    """Return list of app data generated from parser.

    parser : function to handle request
    """
    app_data = []

    # iterate through each row of app_list, confined by start and stop
    for index, row in app_list[start:stop].iterrows():
        print('Current index: {}'.format(index), end='\r')

        appid = row['appid']
        name = row['name']

        # retrive app data for a row, handled by supplied parser, and append to list
        data = parser(appid, name)
        app_data.append(data)

        time.sleep(pause) # prevent overloading api with requests

    return app_data
def process_batches(parser, app_list, download_path, data_filename, index_filename,
                    columns, begin=0, end=-1, batchsize=100, pause=1):
    print('Starting at index {}:\n'.format(begin))

    # by default, process all apps in app_list
    if end == -1:
        end = len(app_list) + 1

    # generate array of batch begin and end points
    batches = np.arange(begin, end, batchsize)
    batches = np.append(batches, end)

    apps_written = 0
    batch_times = []

    for i in range(len(batches) - 1):
        start_time = time.time()

        start = batches[i]
        stop = batches[i+1]

        app_data = get_app_data(start, stop, parser, pause)

        rel_path = os.path.join(download_path, data_filename)
```

```
        # writing app data to file
        with open(rel_path, 'a', newline='', encoding='utf-8') as f:
            writer = csv.DictWriter(f, fieldnames=columns, extrasaction='ignore')

            for j in range(3,0,-1):
                print("\rAbout to write data, don't stop script! ({})".format(j), end='')
                time.sleep(0.5)

            writer.writerows(app_data)
            print('\rExported lines {}-{} to {}.'.format(start, stop-1, data_filename), end=' ')

        apps_written += len(app_data)

        idx_path = os.path.join(download_path, index_filename)

        # writing last index to file
        with open(idx_path, 'w') as f:
            index = stop
            print(index, file=f)

        # logging time taken
        end_time = time.time()
        time_taken = end_time - start_time

        batch_times.append(time_taken)
        mean_time = statistics.mean(batch_times)

        est_remaining = (len(batches) - i - 2) * mean_time

        remaining_td = dt.timedelta(seconds=round(est_remaining))
        time_td = dt.timedelta(seconds=round(time_taken))
        mean_td = dt.timedelta(seconds=round(mean_time))

        print('Batch {} time: {} (avg: {}, remaining: {})'.format(i, time_td, mean_td,
remaining_td))

    print('\nProcessing batches complete. {} apps written'.format(apps_written))
```

Next, we defined some functions to handle and prepare the external files. We defined `get_index` to retrieve the index from a file, maintaining persistence across sessions. This means that every time a batch of information (app data) is written to a file, we write the highest index retrieved within `app_data`. As stated, this is partially for security, ensuring that if there is an error during the download we can read the index from the file and continue from the end of the last successful batch. Keeping track of the index also allows us to pause the download and continue it later.

Finally, the `prepare_data_file` function prepares the CSV file to store the data. If the index we retrieved is 0, it means we are either starting for the first time or starting over. In either case, we want a blank CSV file with only the header row to begin writing to, that is why we wipe the file (by opening it in write mode) and then write the header. Conversely, if the index is anything other than 0, it means that we already had downloaded information on the CSV file and we can leave it alone.

Now we are ready to start downloading data and writing to file. We define our logic particular to handling the steam API - in fact, if no data is returned, we return just the name and appid - then begin setting some parameters. We define the files we will write our data and index to, and the columns for the CSV file. The API does not return every column for every app, so it is best to explicitly mention the column's name.

The data attributes that we retrieved from Steam API are: ['type','name', 'steam_appid','required_age','is_free','controller_support','dlc','detailed_description', 'about_the_game','short_description','fullgame','supported_languages','header_image','website','pc_requirements','mac_requirements','linux_requirements','legal_notice', 'drm_notice','ext_user_account_notice','developers','publishers','demos','price_overview','packages','package_groups','platforms','metacritic','reviews','categories','genres', 'screenshots','movies','recommendations','achievements','release_date','support_info', 'background', 'content_descriptors'].

Here are the first 5 rows of data for some columns in *Figure 5* below:

type	name	steam_appid	required_age	is_free	controller_support	dlc	detailed_description	about_the_game	short_description	fullgame
game	Counter-Strike	10	0	False	NaN	NaN	Play the world's number 1 online action game. ...	Play the world's number 1 online action game. ...	Play the world's number 1 online action game. ...	NaN
game	Team Fortress Classic	20	0	False	NaN	NaN	One of the most popular online action games of...	One of the most popular online action games of...	One of the most popular online action games of...	NaN
game	Day of Defeat	30	0	False	NaN	NaN	Enlist in an intense brand of Axis vs. Allied ...	Enlist in an intense brand of Axis vs. Allied ...	Enlist in an intense brand of Axis vs. Allied ...	NaN
game	Deathmatch Classic	40	0	False	NaN	NaN	Enjoy fast-paced multiplayer gaming with Death...	Enjoy fast-paced multiplayer gaming with Death...	Enjoy fast-paced multiplayer gaming with Death...	NaN
game	Half-Life: Opposing Force	50	0	False	NaN	NaN	Return to the Black Mesa Research Facility as ...	Return to the Black Mesa Research Facility	Return to the Black Mesa Research Facility	NaN

Figure 5. *Sample Data Retrieved from Steam API*

Now let us start by retrieving the data from SteamSpy. by performing a very similar process to the one we used with Steam API. Our parse function is a little simpler here because of the way the data is being returned, and the maximum polling rate of this API is higher so we can set a lower value for `pause` in the `process_batches` function and download more quickly. Apart from that, we set the new variables and make a call to the `process_batches` function once again.

The data attributes that we retrieved from SteamSpy API are: ['appid', 'name','developer','publisher','score_rank','positive','negative','userscore', 'owners','average_forever','average_2weeks','median_forever','median_2weeks','price', 'initialprice','discount','languages','genre', 'ccu', 'tags'].

Here are the first five rows that were retrieved from SteamSpy API, as shown in *Figure 6* below:

appid	name	developer	publisher	score_rank	positive	negative	userscore	owners	average_forever	average_2weeks	median_forever	median_2weeks
10	Counter-Strike	Valve	Valve	NaN	125219	3366	0	20,000,000 .. 50,000,000	11760	1	435	1
20	Team Fortress Classic	Valve	Valve	NaN	3337	634	0	2,000,000 .. 5,000,000	19	0	25	0
30	Day of Defeat	Valve	Valve	NaN	3451	405	0	5,000,000 .. 10,000,000	15	0	9	0
40	Deathmatch Classic	Valve	Valve	NaN	1288	270	0	5,000,000 .. 10,000,000	9	0	12	0
50	Half-Life: Opposing Force	Gearbox Software	Valve	NaN	5296	295	0	5,000,000 .. 10,000,000	381	0	389	0

Figure 6. *Sample Data Retrieved from SteamSpy*

As you might have already noticed there are a lot of attributes that will overlap when merging these two datasets, we will discuss how to solve this issue later.

Now let us talk about the process of retrieving a game's reviews -which is quite simple-, all you need to do is to use the Steam API with the app ID of whatever game you want. We faced an issue during this process where it returned a maximum of 100 reviews per request, so to solve this issue we had to use an important parameter "cursor" According to Andrew Muller's blog "A response includes a 'cursor' attribute, marking which review your request completed on. Including the same cursor in your next request's parameters starts the reviews at the same spot, meaning you get a completely new set of reviews. The cursor is a seemingly random string of characters and may include characters that need to be encoded to work with a URL request.".

The attributes that were retrieved are ['recommendationid', 'author', 'language', 'review', 'timestamp_created','timestamp_updated','voted_up','votes_up','votes_funny', 'weighted_vote_score','comment_count','steam_purchase','received_for_free', 'written_during_early_access','hidden_in_steam_china', 'steam_china_location'].

As you can see a lot of useful attributes were retrieved beside the review text, which gives us the ability to extract meaningful insights that benefit both the user and the developer.

The top 5 rows of the data retrieved using the aforementioned process are shown in the following figure (*figure 7)*:

recommendationid	author	language	review	timestamp_created	timestamp_updated	voted_up	votes_up	votes_funny	weighted_vote_score
78744058	{'steamid': '76561198273957109', 'num_games_ow...	english	doesnt launch with origin\n\nedit: easy anti-c...	1604550435	1637267971	True	6134	912	0.974974
85869577	{'steamid': '76561198142247081', 'num_games_ow...	english	At least you don't have to build an apartment ...	1612340553	1612352988	True	3032	1985	0.969313
82912974	{'steamid': '76561198391638600', 'num_games_ow...	english	Finally I can remove Origin.	1608676590	1608676590	True	1759	631	0.964259
78744672	{'steamid': '76561198020278570', 'num_games_ow...	english	Finally here\n\nIt launches without Origin...\...	1604551696	1604551796	True	2107	91	0.963604
78799524	{'steamid': '76561198233229324', 'num_games_ow...	english	uninstall origin	1604631231	1604631231	True	1177	280	0.962363

Figure 7. *Data Sample of game reviews along with other attributes*

3.1.2 Data Preprocessing Techniques and Cleaning

Data preprocessing and cleaning are essential steps to prepare data for analysis, ensuring the dataset is accurate, consistent, and ready for further processing.

Data cleaning is often referred to as the lengthiest part of any project. So, we will break it up into sections. We will begin by taking care of the game data that we got from the Steam API.

1-Steam API Data

First, we identified missing values, we noticed that some columns have more than 20,000 null values, which means that these attributes will likely not add any meaningful information. So, we designed a function to get rid of any attributes with more than half of their values null.

In the data collection stage, if no information was returned from an app's API request, that means that only the name and appid were stored. We can easily identify these apps by looking at rows with missing data in the `type` column, as all other apps have a value here, as shown in the following figure (*figure 8).*

```
raw_steam_data[raw_steam_data['type'].isnull()].head(3)
```

```
s to remove: 149
```

	type	name	steam_appid	required_age	is_free	controller_support	dlc	detailed_description	about_the_game	short_description	fullgame
6	NaN	Half-Life: Opposing Force	852	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	NaN	Half-Life: Opposing Force	4330	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	NaN	Half-Life: Opposing Force	8740	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Figure 8. *Rows that Contain Nulls in Their Attribute "Type"*

Once we got rid of these rows, we dropped the column "type" as it does not provide any useful information, after that, we also dropped all the rows with null values in the "name" or "steam_appid" columns.

We then got rid of duplicate records because we said earlier that app_id should always be unique. Now we need to process each column on its own, here is an overview of the processing we did to each column.

- **Required_Age**: There are no missing values in this column, but the vast majority have a value of 0. We'll clean the column anyway, but this probably means it won't be of much use in analysis as there is little variance in the data.

- **Platforms**: Ensuring the platforms (like PC, Xbox, etc.) are correctly formatted.

- **Price_Overview & Is_Free**: Whilst checking for null values in the "price_overview" we noticed that most null values could be accounted to the "is_free" column, then if we look at the first row of the data, we can see that there are a variety of formats in which the price is stored.

  Here is an example: "{'currency': 'GBP', 'initial': 719, 'final': 719, 'discount_percent': 0, 'initial_formatted': '', 'final_formatted': '£7.19'}" so we created 2 new attributes one is called "currency", which contains the currency in which the price is being expressed, and the other attribute called "price".

- **Supported_Languages**: After looking at the structure of the columns it looks like the values are stored in all format types from HTML headings to comma-separated values, but since we are interested only in games that support the English language we can check if the game contains the word English in lower case.

- **Developers and Publishers**: Most of the time this attribute contains the same data we removed from the rows that had null values, the values were stored as lists, that could contain one or multiple values, but since most of them contain one value, we decided to add them together and separate them by a comma.

- **Categories and Genres**: After dealing with null values, both of these attributes appear to be a list of dictionaries containing an id and description key-value pair, so extracting the values under the key "description" is what we did.

- **Achievements and Descriptors**: Processing achievements and other descriptors to ensure that they are correctly represented and then exporting them to a different file.

- **Media columns:** Dealing with null values and then exporting them to a different file.

- **Release Date**: Standardizing the release dates to ensure that they are in a consistent and usable format.

2- SteamSpy API

Now it is time to start working on cleaning and preprocessing data that we got from SteamSpy API. Firstly, we noticed that there are a lot of columns here that are similar to the ones we got from Steam API which we had already dealt with -like the column 'name'-, so we decided to use the columns from the other dataset instead. Secondly, we got rid of "score_rank" because it has a lot of null values that add no value, columns that exist in the Steam API dataset, such as "genre", "developer",

"publisher", "initialprice", and others are dropped, and columns that have data that changes rapidly, such as "average_2weeks", "median_2weeks", and "ccu" are removed to focus on more stable attributes. Thirdly, we worked on the 'tag' attribute, by making sure to standardize it and include the top tags that were mentioned only, as shown in the figures below:

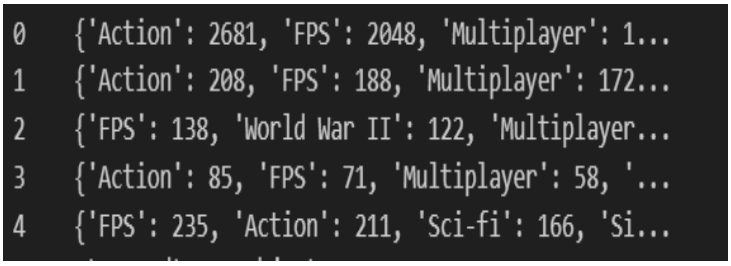

Figure 9. *Game Tags Before Processing*

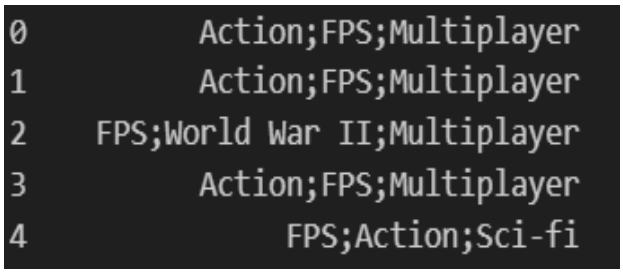

Figure 10. *Game Tags After Processing*

Finally, we reformatted the owners' columns from "100,000 … 300,000" to "100000-300000" so that it looks better.

- **Reviews processing:**

Processing textual data before applying any sentiment analysis techniques is very important since clean and consistent data ensures that sentiment analysis has better accuracy, it also helps with reducing the size of data by removing redundancy and noise.

Firstly, we got rid of noise like HTML headers or anything that does not represent a word. Secondly, we standardized the text by using the function ".lower()" on it. Thirdly we lemmatized the text so that similar words would be treated as the same word. Fourthly, we tokenized the text, which is done by dividing a sentence or a phrase into smaller units. Finally, we got rid of stop words like "the/and/or" to reduce the dimensionality.

3.1.3 Data Integration and Transformation

To combine the two datasets from Steam and SteamSpy we merged them using inner join to keep rows that only exist in both datasets.

In the end, we removed some columns that overlapped like 'name_steamspy', 'steam_appid', etc., and we also renamed several columns.

3.2 Feature Engineering

After deciding the attributes we wanted to include in the process of building the recommendation system, which was (‘**categories**’,’**genres**’,’**steamspy_tags**’, ’**publisher**’) we needed to combine these attributes and make sure that they were in the right format so that we could use them as an input to create TF-IDF matrix, then after that, we could use the TF-IDF matrix to calculate the cosine similarity between games. We needed to make sure that there was a blank space between each word in the TF-IDF input and then standardize words like “multi-player” and “multiplayer”. An example of how the TF-IDF_input attribute would look like is shown in the following figure:

genres	steamspy_tags	publisher	categories
Action	Action;FPS;Multiplayer	Valve	Multi-player;Online Multi-Player;Local Multi-P...
Action	Action;FPS;Multiplayer	Valve	Multi-player;Online Multi-Player;Local Multi-P...
Action	FPS;World War II;Multiplayer	Valve	Multi-player;Valve Anti-Cheat enabled
Action	Action;FPS;Multiplayer	Valve	Multi-player;Online Multi-Player;Local Multi-P...
Action	FPS;Action;Sci-fi	Valve	Single-player;Multi-player;Valve Anti-Cheat en...

Figure 11. *Attributes that will Be Inputs For TF-IDF*

tfidf_input
Action Multiplayer OnlineMultiPlayer LocalMult...
Action Multiplayer OnlineMultiPlayer LocalMult...
Action Multiplayer ValveAntiCheatenabled Valve...
Action Multiplayer OnlineMultiPlayer LocalMult...
Action Singleplayer Multiplayer ValveAntiCheat...

Figure 12. TF_IDF Inputs

Chapter 4: MODEL DEVELOPMENT AND ARCHITECTURAL DESIGN

This chapter outlines the development and architectural design of the platform, which integrates a game recommendation system, sentiment analysis, and an AI-powered chatbot. The recommendation system uses advanced algorithms to provide users with personalized game suggestions based on their preferences, addressing the challenge of navigating Steam's vast library. Sentiment analysis processes user reviews to offer valuable insights into player feedback, highlighting strengths and areas for improvement. The chatbot adds an interactive dimension, allowing users to receive tailored game recommendations through conversational inputs. Together, these components form a comprehensive solution to enhance user experience and assist developers in understanding their audience.

4.1 System Architecture Overview

Our platform has three main parts: a game recommendation system, a sentiment analysis tool, and a chatbot. These parts work together to help Steam users find games and give developers insights into player feedback.

The recommendation system uses advanced methods to study what users like and suggests games they might enjoy. It can handle large amounts of data from Steam and is designed to grow as more users join. The sentiment analysis tool looks at reviews from players, using language processing to find out what people like and don’t like about games. This helps developers understand their audience better.

The chatbot [7] is an interactive feature that makes it easier for users to find games. Users can describe what they want, and the chatbot suggests games that match their description.

The system is designed to be easy to update and maintain. Each part can be improved on its own while still working well with the others. It is built to handle more users in the future and to keep working even if something goes wrong. This design makes the platform helpful and reliable for both gamers and developers. As shown in the figure below:

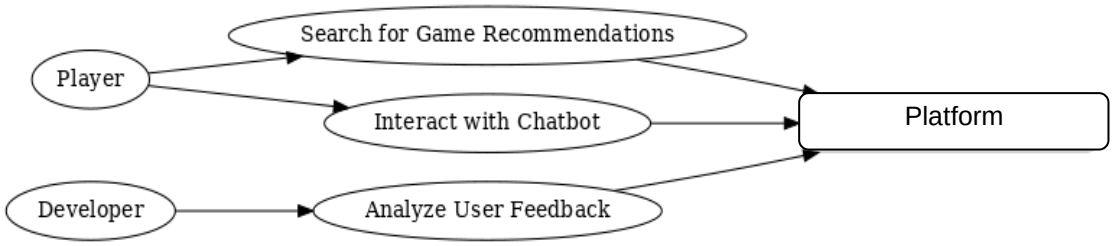

Figure 13. System Architecture Overview

4.2 Data Pipeline Design

The data pipeline in our project is designed to handle the flow of game-related information efficiently, from collection and processing to storage and usage.

- **Data Collection**

The pipeline starts with collecting game details, user reviews, and ratings, all stored in a structured CSV file. This file serves as the primary source of information for the system.

- **Data Processing**

Collected data undergoes preprocessing to ensure it is clean and organized. Features like game genres, ratings, and tags are extracted for the recommendation system. User reviews are analyzed using Natural Language Processing (NLP) techniques to identify sentiments, enabling the sentiment analysis component to provide meaningful insights.

- **Data Storage**

The processed data is stored in a vector database, where embeddings are created from game descriptions and reviews. These embeddings enable quick and precise similarity searches, which are crucial for delivering accurate recommendations and supporting the chatbot functionality.

- **Data Usage**

When users interact with the system, the stored data is accessed to generate game recommendations based on their preferences. The chatbot uses this data to suggest games that match user inputs, enhancing the overall user experience.

4.3 Model Architecture and Design

The models used in our project were designed to support the recommendation system, sentiment analysis, and chatbot features. Each model was chosen and built to be efficient, accurate, and easy to use.

- **Recommendation System**

The recommendation system uses a method that matches users' preferences with game data. By analyzing game information like genres and tags, the system finds games similar to what the user likes. It ensures personalized and accurate recommendations by comparing the features of games using vectorized data. As shown in the figure below:

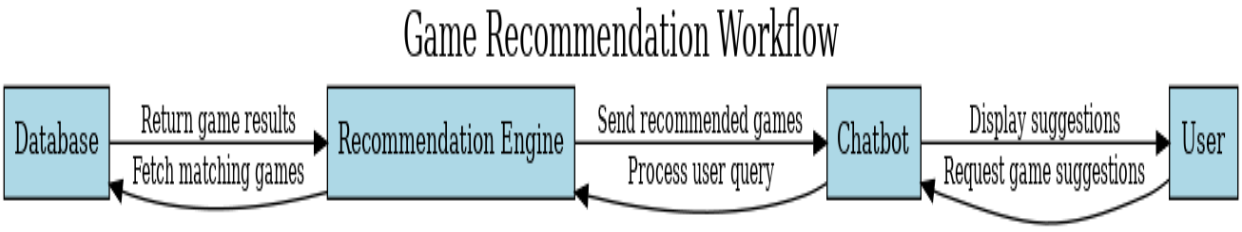

Figure 14. Game Recommendation Workflow

The sentiment analysis feature processes game reviews to determine if users' feedback is positive, negative, or neutral. This helps developers understand how players feel about their games and identify what they are doing well or need to improve. As shown in the figure below:

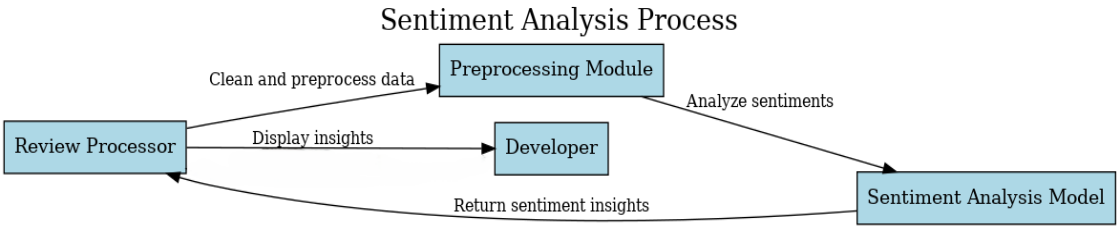

Figure 15. Sentiment Analysis Process

- **Chatbot Design**

The chatbot is designed to recommend games based on user input. It takes user descriptions of the type of game they want and provides suggestions from the recommendation system. The chatbot makes it easy for users to interact with the platform and find games they'll enjoy. As shown in the following figure:

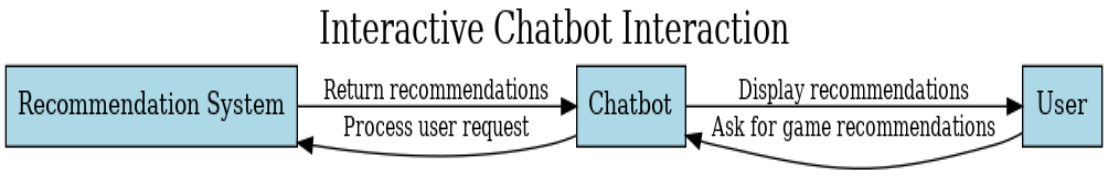

Figure 16. Interactive Chatbot Interaction

- **Simplicity and Scalability**

The models are designed to provide clear and easy-to-understand outputs, like game suggestions or sentiment scores. They can also handle more data as the platform grows, ensuring the system remains fast and efficient over time.

- **Tools and Frameworks**

The project uses reliable tools and libraries to process data, store it, and deliver results quickly. These tools make the models work smoothly and allow for future updates if needed.

By combining these features, the models in the system work together to create a helpful and efficient platform for users and developers.

❖ The functional requirements define the website's core features, such as game recommendations, sentiment analysis, and a chatbot for user interaction. Non-functional requirements specify quality attributes like usability, performance, scalability, and reliability. Together, they ensure the platform meets user needs and operates efficiently.

The **functional requirements** of our website are:

- The user shall be able to search for recommendations using the game's name
- The user shall be able to click the recommended game name, which in return will direct the user to the game store page
- The user shall be able to check the game statistics, game review analysis, most frequent words used in reviews, and sentiment analysis by topic
- The user shall be able to ask the website chatbot for game recommendations based on their preferences
- The user shall be able to ask the website chatbot to give them an overview and a description based on a game title

- The user shall be able to check the game reviews by asking the website chatbot
- The user shall be able to ask the website chatbot for players insight and suggestions to improve and address the game mechanics and features

The **non-functional requirements** of our website are:

- The website shall provide a user-friendly interface that allows users to interact with the recommendation system, sentiment analysis, and chatbot effortlessly.
- The website shall be designed to handle an increasing number of users and data volumes without significant degradation in performance.
- The website shall allow for the integration of additional features or modules in the future.
- The website shall process user queries and provide recommendations or analysis results within an appropriate time.
- The website shall maintain rapid response time for chatbot interactions under normal operating conditions.
- The website shall achieve 99.9% uptime, ensuring availability for users at all times.

4.4 Code Implementation

The code implementation translates the project's architecture into a functional system, integrating the recommendation system, sentiment analysis, and chatbot. It emphasizes modularity and scalability for future enhancements. Key components and their implementation details are outlined below:

- **Game Recommendation System**

  The recommendation system is built using data processing and analysis techniques to suggest games based on user preferences. The system leverages preprocessed data from a CSV file containing game details, user ratings, and reviews. Algorithms for similarity and relevance scoring were used to identify games that align with user preferences. The system ensures quick and accurate recommendations using efficient data handling and querying methods.

- **Sentiment Analysis Module**

  The sentiment analysis component analyzes user reviews to extract insights about game performance, story, and gameplay. Natural Language Processing (NLP) techniques were used to classify reviews into positive, negative, or neutral sentiments. This helps developers understand user feedback more effectively and address areas of improvement.

- **Chatbot for Game Recommendations**

  The chatbot provides a conversational interface for users to find games they like. It processes user inputs to understand preferences or descriptions of desired games and suggests relevant titles. The chatbot integrates the recommendation system to deliver personalized and accurate results in real-time.

- **Integration of Components**

  A pipeline was established to connect the recommendation system, sentiment analysis module, and chatbot. Data is processed and passed seamlessly between these components to maintain system coherence.

Chapter 5: MODEL/SYSTEM EVALUATION AND TESTING PLAN

5.1 Testing Plan Overview

We plan to test the entire architecture of our system, beginning with the recommendation system, followed by the text analysis and the RAG pipeline, to ensure that all components function seamlessly in real-time scenarios.

5.1.1 Unit Testing

The code was structured with a clear and modular approach, where each function was designed to perform a single, specific task. This design choice simplifies the process of testing individual functionalities, ensuring each component can be evaluated independently for accuracy and reliability.

We began by testing the recommendation system, which serves as the foundation of our project. To ensure its functionality, we evaluated the system using a large dataset of games. The system was tested to generate ten recommendations for each game in the dataset, verifying that the suggested games were relevant and aligned with the content-based recommendation approach implemented. Once we confirmed its effectiveness, we proceeded to focus on the next phase of the project.

Next, we shifted our focus to the text analysis component, which required significant effort due to its complexity and reliance on multiple distinct techniques. Each technique handled different aspects of the data, so careful integration was required to ensure the module worked well as a whole. We focused on synchronization by testing each technique both individually and as part of the entire system, resolving any issues to maintain the accuracy and reliability of the text analysis process. Below, we will explain how we ensured each technique functioned properly:

- **Sentiment and statistical Analysis**: Sentiment classification was validated using labeled datasets by comparing the predicted sentiment scores with ground truth labels. These results were then utilized to create visualizations, such as a pie chart displaying the percentage of players who liked or disliked a game. Additionally, statistical analyses were conducted to ensure accuracy, including calculating the mean and median playtime for each game. This comprehensive approach ensured both the reliability of the sentiment analysis and the statistical insights derived from the data.
- **Word cloud** : Creating the word cloud relied heavily on user reviews, which often contained noise. To address this, we began by cleaning the reviews, removing any unusual symbols, irrelevant text, and stop words, as these could impact the final results. We then applied normalization and lemmatization techniques to ensure that words with slight variations were not treated as different words. This process ensured that the word cloud accurately represented the most significant terms from the reviews.
- **Semantic by topic** : Semantic analysis by topic presented a significant challenge, as it required not only determining the polarity score of a review but also identifying the specific topic it addressed. For the purpose of our project, we narrowed the focus to three key topics: performance, story, and gameplay. Each review was analyzed to determine its relevance to these topics by checking for the presence of associated keywords. Based on the polarity score, we then classified the review as discussing the topic in either a positive or negative context. This method ensured a targeted and meaningful analysis of user feedback.

Next, we focused on testing the RAG (Retrieval-Augmented Generation) pipeline, which is a key part of delivering accurate and contextually relevant recommendations and responses. The process involved carefully evaluating each stage to ensure everything worked as expected:

- **Data Retrieval**:

We started by testing how well the pipeline could fetch relevant data from the database based on user queries. This included verifying that the Pinecone vector database consistently returned accurate and relevant results.

- **Query Refinement**:

To handle ambiguous or incomplete inputs, we tested the query refinement step. This was done to make sure it could effectively use the conversation history to clarify and refine user queries.

- **Context Integration**:

We ensured that the retrieved data was correctly integrated into the response generation process, allowing the system to provide answers that were both coherent and directly relevant to the user's input.

- **Response Quality**:

The quality of the responses was assessed to ensure they were accurate, relevant, and natural. We also tested various edge cases, like incomplete or conflicting information, to make sure the system handled them gracefully.

- **Error Handling**:

We tested scenarios where the retriever returned no results or incomplete data to confirm that the system could still respond appropriately without interrupting the user experience.

- **Performance Testing**:

Finally, we measured how the pipeline performed under different conditions, checking for delays and making sure it could handle multiple queries at the same time.

Through this testing process, we made sure the RAG pipeline was both reliable and efficient, forming a strong foundation for the system's ability to provide recommendations and respond to user queries about any game in vector database.

5.1.2 Integration Testing

Integration testing was conducted to ensure that all components of the system, including the web application built using Streamlit [8], worked together seamlessly to provide cohesive and functional user experience. This phase focused on validating the interaction between the recommendation engine, sentiment analysis module, RAG pipeline, and the user interface, ensuring data consistency and reliable performance throughout the system.

Testing Approaches:

- **Module Interaction**:

  We began by validating the interaction between core modules, such as the recommendation system, text analysis components, and the RAG pipeline. The outputs from one module were tested to ensure they were compatible with the inputs of the subsequent module. For example, the recommendations generated by the content-based filtering approach were passed to the user interface for display and further interaction.

- **API and Database Integration**:

  The system relied on APIs and the Pinecone vector database for data retrieval and processing. Integration tests confirmed that data fetched through APIs, including game information and user reviews, were correctly stored, processed, and displayed within the Streamlit interface. The Pinecone database was verified to return relevant and accurate data for user queries.

- **Streamlit User Interface Testing**:

  The integration of the Streamlit application was tested extensively. Each feature, including the game recommendation display, sentiment analysis visualizations, word clouds, and semantic topic breakdowns, was evaluated to ensure data was accurately represented in a user-friendly manner. Tabs and input fields were tested for responsiveness and usability.

- **End-to-End Workflow**:

  Comprehensive end-to-end testing scenarios were executed to mimic real-world user interactions. For example, a user could input a query in the chatbot, receive personalized recommendations, and view visualizations and analysis of game reviews seamlessly within the application.

Key Scenarios Tested:

- **Data Flow Between Components**:

  Inputs from the Streamlit UI were tested to ensure they were correctly processed by the backend, and results were accurately displayed. For instance, a user query in the chatbot triggered the RAG pipeline, which retrieved data and generated responses displayed within the chat interface.

- **Error Handling and Recovery**:

  Integration scenarios included handling cases where the recommendation engine or sentiment analysis returned incomplete or no results. The system's fallback mechanisms, such as displaying an informative error message or providing default recommendations, were validated.

- **Visualization Accuracy**:

  Charts, word clouds, and other visualizations were tested to ensure they correctly reflected the processed data. For example, the pie charts for sentiment analysis and word clouds derived from reviews were cross-verified with raw data for accuracy.

- **System Performance and Latency**:

  The system's responsiveness was measured during integration testing to ensure minimal delays when switching between Streamlit tabs or retrieving data from the RAG pipeline. The system successfully handled multiple simultaneous user queries without significant latency.

❖ **Results and Observations:**

The integration testing confirmed that all modules and the Streamlit web application functioned harmoniously. Data flowed seamlessly between components, and the user interface provided a smooth and interactive experience. Error handling mechanisms ensured system reliability, and performance testing validated the application's scalability and responsiveness under different conditions. This robust integration established the platform as a reliable and efficient tool for game recommendations and sentiment analysis.

5.2 System Evaluation

The system evaluation focused on examining the performance and reliability of the developed platform under various operational scenarios. This included assessing its scalability and responsiveness to user inputs. Each aspect was tested to ensure the platform meets the expected standards for a smooth user experience.

5.2.1 Scalability Assessment

The system's scalability was assessed by evaluating how well its components functioned under typical operating conditions. The recommendation system, sentiment analysis module, and the Streamlit application were tested to ensure consistent performance. Key observations included:

- **Database Queries**: The Pinecone vector database consistently delivered fast and accurate results during retrieval, confirming its ability to support efficient operations.
- **System Stability**: All components maintained functionality and efficiency when tested under standard workloads, ensuring reliable performance.

5.2.2 Responsiveness and Latency

Responsiveness and latency were critical metrics to evaluate the platform's ability to deliver a seamless experience. Various user interactions were timed to measure response speed, with particular focus on:

- **Recommendation Retrieval**: The recommendation system generated results within a few seconds, ensuring timely feedback on user queries.
- **User Interface Interactions**: The Streamlit interface responded quickly to inputs, such as switching between tabs, submitting queries, and displaying visualizations like sentiment charts and word clouds.
- **System Resilience**: Even during more complex operations, such as processing detailed sentiment analysis or generating semantic visualizations, the system maintained acceptable response times.

Chapter 6: SYSTEM DEPLOYMENT AND INTEGRATION

This chapter outlines the process of deploying and integrating the system into its operational environment. It covers the steps taken to ensure the seamless implementation of the system's components, the tools and technologies utilized, and the strategies employed for integration. The aim was to create a functional, user-friendly platform capable of delivering efficient recommendations and meaningful analysis.

6.1 Deployment Process

The deployment of the system focused on ensuring accessibility, stability, and ease of use for end users. The Streamlit framework was chosen for its lightweight nature and interactive capabilities, allowing for the rapid development and deployment of a web-based application.

- **Environment Setup:**

The application was deployed on a cloud-based server to ensure availability and scalability. Virtual environments were created to manage dependencies and prevent conflicts between packages.

The Pinecone vector database was hosted separately to manage the RAG pipeline’s backend efficiently.

- **Hosting Platform**:

The web application was deployed using a reliable hosting service that supports Streamlit, ensuring a seamless user experience.

- **Security Considerations**:

API keys and sensitive configurations were securely stored using environment variables to protect the system from unauthorized access.

6.2 System Integration

The integration process ensured that all components of the system worked together cohesively. This included combining the recommendation engine, sentiment analysis module, and user interface into a unified platform.

- **Backend and Frontend Integration**:

The recommendation system and sentiment analysis module were integrated with the Streamlit interface to provide real-time responses to user inputs. Outputs from the backend were formatted and displayed interactively on the frontend.

- **Database Connectivity**:

The Pinecone vector database was linked to the backend, enabling efficient retrieval of game data and similarity scores. Queries to the database were optimized to minimize latency.

- **Data Flow Coordination**:

Data inputs, processing, and outputs were coordinated to ensure consistency. For example, sentiment analysis results were passed directly to the visualization module to generate pie charts and word clouds.

- **Error Handling**:

Mechanisms were implemented to handle errors gracefully. For instance, fallback responses were provided if the recommendation system or database retrieval failed, maintaining a smooth user experience.

6.3 User Interface Design

The user interface was designed to be intuitive and accessible, providing users with clear pathways to interact with the system's features. Key considerations included:

- **Navigation**: Tabs and input fields were structured to guide users through recommendations, sentiment analysis, and visualizations.

- **Interactivity**: The chatbot allowed users to enter queries and receive instant feedback, ensuring engagement and ease of use.

- **Visualization**: Interactive graphs and charts were used to display sentiment analysis results, enhancing the user experience with clear and meaningful representations.

6.4 Deployment Challenges and Resolutions

During deployment, several challenges were encountered and resolved:

1. **Latency Issues**:

Optimizations were made to the RAG pipeline and database queries to reduce response times.

2. **Dependency Conflicts**:

Virtual environments were configured to isolate dependencies, preventing conflicts between packages.

3. **API Limitations**:

Rate-limiting issues with external APIs were addressed by implementing caching mechanisms and retry logic.

Chapter 7: CONCLUSION AND FUTURE WORK

This chapter provides a summary of the project's accomplishments, evaluates its outcomes against the initial objectives, and outlines potential future enhancements to build upon the foundation established in this work.

7.1 Summary and Project Evaluation

The primary goal of this project was to develop a platform that combines a recommendation system with sentiment analysis to enhance the user experience and provide actionable insights for game developers. Throughout the project, we successfully achieved the following:

- **Recommendation System**:

A content-based recommendation engine was developed and tested, effectively providing personalized game suggestions based on user input and data.

- **Sentiment Analysis**:

We implemented a sentiment analysis module capable of extracting meaningful insights from user reviews. This included visualizations like word clouds, sentiment polarity charts, and topic-based analysis.

- **Statistical Analysis**:

Statistical analysis was conducted to gain insights into games and their player bases. This included metrics such as the number of people who own a game, the average and median playtime of players, and the ratio of positive to negative reviews. These analyses provided a deeper understanding of user behavior and game performance.

- **Interactive Chatbot**

Developed a conversational AI chatbot to simplify the recommendation process. The chatbot allows users to describe the type of game they are looking for and provides tailored suggestions based on their input. By integrating with the recommendation system, the chatbot enhances user interaction, offering a more personalized and engaging experience.

- **User-Friendly Interface**:

The application was deployed using Streamlit, providing an interactive and accessible interface for users to explore recommendations and review analyses.

- **Robust Testing and Integration**:

Extensive testing ensured that all components, including the RAG pipeline, recommendation engine, sentiment analysis module, and statistical analysis, functioned cohesively.

The project successfully addressed the challenges outlined in the problem statement, including the overwhelming volume of games and reviews on platforms like Steam. By combining advanced data processing with user-centric design, the platform delivers value to both gamers and developers. While the system meets its primary objectives, there are opportunities for further development and refinement.

7.2 Future Directions

Building on the current achievements, several potential enhancements can be explored to expand the system’s capabilities:

- **Enhanced Recommendation Techniques**:

Incorporate hybrid recommendation methods by combining content-based filtering with collaborative filtering to improve recommendation accuracy and diversity.

Introduce user behavior tracking to refine recommendations dynamically based on interaction history.

- **Expanded Sentiment Analysis**:

Extend topic-based sentiment analysis to include additional categories, such as pricing, graphics, or multiplayer experience.

Utilize advanced NLP models for deeper contextual understanding and more precise sentiment classification.

- **Advanced Statistical Analysis**:

Include more detailed insights, such as trends in player engagement over time, comparative analysis of player bases for similar games, and correlations between game features and player satisfaction.

- **Scalability Improvements**:

Optimize the system to handle larger datasets and more concurrent users, ensuring performance remains consistent as the platform grows.

- **Gamification and Personalization**:

Introduce features like personalized dashboards and gamified elements to increase user engagement and retention.

- **Developer-Centric Features**:

Provide additional insights for developers, such as trend analysis over time or benchmarking against competitor games.

- **Multi-Platform Support**:

Extend the system to include games from other platforms, such as PlayStation or Xbox, to cater to a broader audience.

- **Deployment Enhancements**:

Transition to a more scalable and secure deployment infrastructure, such as containerized applications using Docker or Kubernetes.

REFERENCES

[1] https://partner.steamgames.com/doc/webapi || *by* ***Valve*** ||

Access date (26/Mar/2024 - 4:10 pm)

[2] https://steamspy.com/about || *by* ***Sergey Galyonkin*** ||

Access date (26/Mar/2024 – 6:30 pm)

[3] https://steamapi.xpaw.me/# || *by* ***Pavel Djundik*** ||

Access date (26/Mar/2024 – 9:00 pm)

[4] Archives | Nik Davis (https://nik-davis.github.io/tag/steam.html) || *by* ***Nik Davis*** ||

Access date (27/Mar/2024 - 12:15 pm)

[5] Scraping Steam User Reviews. Using python to scrape user reviews… | by Andrew Muller | Medium (https://andrew-muller.medium.com/scraping-steam-user-reviews-9a43f9e38c92) || *by* ***Andrew Muller*** ||

Access date (14/Apr/2024 – 4:00 pm)

[6] https://steamspy.com/api.php || by **Sergey** ||

Access date (16/Apr/2024 – 6:00 pm)

[7]https://blog.futuresmart.ai/building-an-interactive-chatbot-with-langchain-chatgpt-pinecone-and-streamlit || *by* **Pradip Nichite** ||

Access date (1/Sep/2024 - 6:15 pm)

[8] https://docs.streamlit.io/ || *by* ***Streamlit*** ||

Access date (15/Nov/2024 - 3:00 pm)