Question-Answering System for Game Reviews

Prathik Srinivasan
ps94@illinois.edu
University of Illinois Urbana-Champaign
Champaign, Illinois, USA

Abstract

ACM Reference Format:
Prathik Srinivasan. 2025. Question-Answering System for Game Reviews. In . ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

The goal of this project is to develop a retrieval-augmented generation system (RAG) system to analyze game reviews on the popular PC game distribution platform, Steam. We will initially highlight the motivation and significance of this project, comparing it to other similar tools to indicate the novelty of the solution. Following that, we will describe the implementation process, with a detailed overview of the overall system architecture as well as a more specific look at the retrieval process and prompt construction. Finally we showcase the results of the evaluation of this tool, conducted through a user survey, and highlight potential avenues for future expansion. **The code for this tool can be found at https://github.com/prathiksrinivasan/cs510-g24**

2 Project Motivation and Significance

2.1 Steam User Reviews

Steam is the largest PC game distribution platform, with over 130 million monthly active users on the platform, and nearly 70 million users on a daily basis. [5] The platform has hosted over 100,000 games since 2007, with the amount of games being released on the platform increasing year over year. In 2024 alone, over 18,800 games released on the platform, and to date over 6800 games have been released in 2025. [2] For the average consumer, filtering through all or even a subset of these releases is a near-impossible task. Over nearly two decades, Steam has developed an extremely robust recommendation algorithm, utilizing data from user play times, affinity for genres, and click-through rates to push users towards games that they may be interested in and separate the wheat from the chaff. The most powerful tool on the platform, however, is the system of user-generated reviews. For any game in their library, users are given the option to write a review of any length, marking it as "recommended" or "not recommended". This aggregate score is the most visible and transparent aspect of Steam's recommendation system, with review amount generally correlating to sales and aggregate score thresholds being indicative of Steam's tendency to promote the game.

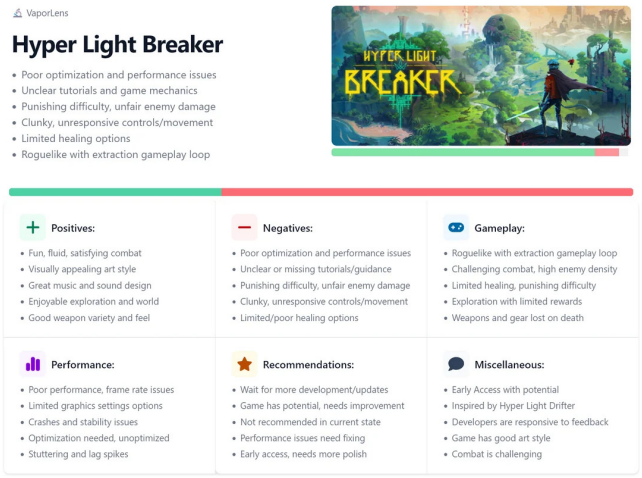

Figure 1: ***A screenshot of a similar review aggregator and summarizer known as "VaporLens" Although this summarizer highlights certain relevant topics that a customer may be interested in, there is no functionality for the user to seek additional information or view the text of reviews directly relevant to their query.***

2.2 Benefit to Users and Developers

Unfortunately, despite the robust user review system and the willingness of the Steam user base to engage with writing informative reviews, browsing the user reviews themselves is a difficult task. If a customer wants to glean more in-depth information about a specific title, they will have trouble finding any specific relevant content. Although reviews can be filtered by helpfulness and rating, there is no search functionality, semantic or otherwise, that users can utilize to find reviews that pertain to information they are interested in. A retrieval system that allows users to search and filter Steam user reviews would be beneficial to any prospective customer of a steam game, as well as developers conducting market research or evaluating user feedback. If a user's system is prone to specific issues or if they have concern about a specific prevalent issues, they would be able to identify reviews that hold relevant info to understand the general sentiment in relation to specific aspects of a game's gameplay, audio, artwork, narrative, performance, etc. For developers, such a tool would be invaluable in conducting market research. For independent game developers specifically, the best indication of whether a game can succeed or fail is the volume and

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

content of user reviews. Currently, if any developer wants to evaluate some game with similarities to their own, they must manually look through reviews to understand what players may have liked and disliked. With a semantic retrieval tool, this process could be simplified to filter relevant information, which would be especially helpful for researching the successes and failures of games with a large volume of reviews. Developers would be able to apply this benefit to their own games as well, gaining a better understanding of their player base with a streamlined process of understanding user reviews.

2.3 Comparison of Existing Solutions

I am hardly the first person to come up with the concept of utilizing LLM's capability to quickly evaluate large amounts of data to simplify the task of sifting through user reviews in general, nor the first to apply this concept to Steam user reviews specifically. <citations> In my research, I found several websites that allow you to search for a steam game, using GPT or some other language model to summarize a subset of the reviews that it collects and create some sort of recommendation. [3] [4] There is a clear demand for a tool like this as well, with a post on the r/Steam subreddit about VaporLens, one such AI review aggregator, collecting 1.5k upvotes. [6] However, all of these existing solutions, while differing slightly in the quality and content of generated summaries, generally miss the underlying need that a tool such as this should aim to address. While there is some value in aggregating and summarizing the general sentiment surrounding the game, ultimately the end result of such a summary is effectively serving the same purpose as Steam's own recommendation system itself and the aggregate rating. If a user has specific questions that are not addressed in the summary, they will have to manually look through the reviews to find that information. This solution seeks to address this challenge, combining a semantic information retrieval system with large language models to create a question-answering chat interface that can answer specific user concerns.

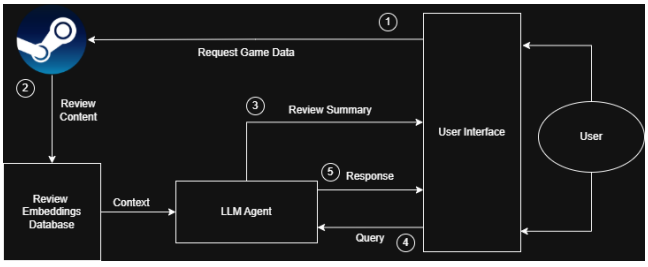

Figure 2: ***Diagram of the overall user flow of the application. (1) Game data is requested through user interface. (2) Review content is retrieved and processed into vector embeddings (3) Review text is used as context for the LLM agent to generate a summary. (4) A user query is retrieved and transformed into a vectorized embedding. (5) A response is generated by the LLM using context retrieved based on the query.***

3 System Architecture

The solution I developed takes the form of a browser-based application with a chat-like interface. After identifying the Steam product that they want to analyze, the user can utilize natural language queries to receive answers to their questions. The responses to the queries will also be in a natural, conversational language and draw their information directly from user reviews of the product. With this combination of question-answering behavior and conversational interface with text generated by a RAG system, I can address the issue of finding specific relevant information while maintaining the usability of LLM chat interfaces.

3.1 Overview

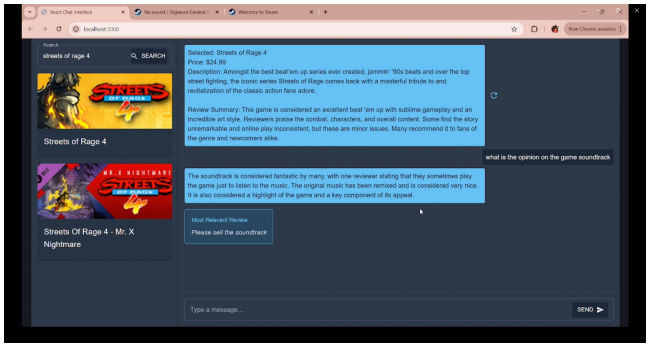

Figure 3: ***A showcase of the user interface of the application, including the game search area (left), and chat interface (right) with a generated summary, sample query and response, and retrieved relevant review.***

3.1.1 User Interface. The user interface of the application mirrors chat interfaces on various LLM powered tools, such as ChatGPT, Gemini, DeepSeek, etc. A section on the left-hand side of the page allows the user to search for products by name or 'App ID' with the results from the Steam API's retrieval of closest matches appearing in the column. Once an item is selected from that list, a request is made to the Steam API and the price, description, and a subset of reviews are retrieved. Any information generated by the back-end LLM model is output to the main chat portion of the screen with the appearance of a message, and once the information is loaded the user can enter their own queries to find information about specific topics. The front-end design is developed with ReactJS and connected to a Python back-end via a set of Python Flask API calls.

3.1.2 Data Requesting and Storage. As described previously, once a user selects a specific item, the information for that app ID is requested from the Steam API endpoint. The initial request contains general information about the product itself, including the title, price, and description. This is returned in JSON format. After this, using the app ID, a different API endpoint is used to request reviews. These are returned in batches of 20 reviews by default, but for the purposes of this prototype we increase this to the maximum of 100 reviews. These reviews are also returned in JSON format, containing the reviewer's information, whether or not they recommended the product, and the review text itself. Using LangChain's document

text splitting and integration with google's VertexAI embeddings, we parse the raw review text for each review and convert it to a vectorized embedding. This text and associated embedding is stored in a LangChain InMemoryVectorStore structure. The decision to use this over a more robust database system like LangChain's ChromaDB integration was because this data is only intended to be stored as long as the user is evaluating a given application. When the user retrieves data from a new title, the previous information is discarded and replaced with new metadata and reviews.

3.1.3 RAG system. Once the data has been requested and processed from the Steam API, the system utilizes a subset of the collected reviews as context when generating a response for the user. Initially, 10 reviews are selected to generate an overall summary of the user sentiment, which is displayed to the user alongside the price and description of the selected title. When a query is recieved from the user interface, the prompt is utilized to retrieve relevant documents (A process that will be elaborated on in section 3.2) and the LLM utilizes these as context to generate an answer. The language model system that we use by default is Google Gemini, specifically the 2.0-flash model. Other language models would work as well with the appropriate LangChain libraries applied. Google Gemini was selected for the easy access to free computational resources.

3.2 Document Retrieval Process

To retrieve relevant documents to the user query, several steps are applied. First, the user query is converted into a vectorized embedding representation using Google's VertexAI system, using the same embedding model as the review text. This embedding is then compared against the elements in the vector store database, finding the vector embeddings with the highest cosine similarity to the query vector. The associated review text for the highest similarity score vectors are then collected and passed to the prompt context. For the initial summary of user reviews, a 'dummy' query is used to collect a random set of 10 reviews from the database as a representation of the entire set. This number was selected to minimize context length while still having a diverse set of information. For any user query, the 5 most relevant reviews are collected from the database. In addition to this, 3 more reviews are randomly selected to the database for incorporation into the prompt context. The reasoning behind this choice of including random irrelevant reviews is detailed in the paper *The Power of Noise: Redefining Retrieval for RAG Systems.* [1] The authors of this paper experimentally find that augmenting the context of a RAG system with random documents appears to focus the attention of the LLM on the relevant documents and results in higher accuracy levels.

3.3 Prompt Construction

Once the relevant text is retrieved for the user query, the raw text is compiled and passed to the language model as a text prompt. The prompt structure itself is retrieved from LangSmith and populated with the relevant information for each query. The general format of the prompt construction provides a system prompt with instructions, the text of the user query (if applicable) and the context, which is given as a list of the original review texts. For the initial summary, the prompt differs slightly and does not include a query.

The structure of the initial summary prompt is as follows:

Average Relevance	4.389
Average Helpfulness	4.178
Average Coherence	4.711

Table 1: Results of User Survey on RAG system response accuracy

> "You are an assistant for generating a summary related to the reviews of online videogames. Use the following pieces of retrieved context in the form of user reviews to create a generalized summary of user perceptions of this game. Do not use any additional information to generate an answer. Use four sentences maximum and keep the answer concise.
> Context: {context}
> Answer:"

The LLM will fill in its response following "answer", which can then be parsed and returned to the front-end to be displayed in the chat interface. in the {context} segment, the raw text from each review is added as n: <review text> where n is the number of the review.

For responses to user queries, we utilize the following prompt which has a similar structure but changes the task description and incorporates the user query text in the {question} section:

> "You are an assistant for question-answering tasks related to the reviews of online videogames. Use the following pieces of retrieved context in the form of user reviews to answer the question. Do not use any additional information to generate an answer. If you don't know the answer, return "Unable to find answer in retrieved reviews" with no additional text. Use three sentences maximum and keep the answer concise.
> Question: question
> Context: context
> Answer:"

4 Evaluation

I evaluated my system on two benchmarks, intended to address overall functionality and robustness to specific types of interactions. The benchmarks themselves were designed specifically for this project, as I felt it was more important to address the specific usability of this tool in a vaccuum, due to its novel application.

4.1 Accuracy Benchmark

The primary evaluation benchmark was for the "accuracy" of the generative answers to questions as evaluated by users. I decided to focus on the user evaluation of these summaries rather than a quantitative measurement of retrieval relevance, due to the user-focused quality of this solution and the lack of a quantitative dataset on which to evaluate. To conduct the user survey, I first constructed 3 possible examples of queries that users may supply to the tool. The questions were "Is the soundtrack of the game good?", "How fun is the game after 5 hours?" and "Does the game have performance issues?". All of these questions are representative of common topics that Steam consumers may care about and search through reviews

to learn about. For each of these queries, I utilized my pipeline to retrieve answers for 5 different games, with varied genres, review amount, scores, etc. I created a survey in which respondents would rank each response to a query on a Likert scale from 1 to 5, for 3 different categories of accuracy. The relevance of the generated response to the question, the helpfulness of the response in answering the question, and the coherence of the response text. On the Likert scale, 1 would be not relevant/helpful/coherent and 5 would be very relevant/helpful/coherent. I compiled these sample queries and responses into a survey and collected data from 12 respondents. The results can be seen in Table 1. As we see from the results, the area in which the system performs the best is coherence, primarily due to the power of Google Gemini's language model. In general, the system appears to retrieve and generate text that is relevant to the question, but users observed a tendency to incorporate additional information from retrieved reviews within the response. It is likely to me that this issue could be solved with further prompt engineering, rather than adjusting the retrieval method. It would be difficult to separate only relevant information out of the source documents, as steam reviews often cover multiple topics. In regards to helpfulness, measuring this metric may have been a misguided choice as there is no guarantee of a helpful review within the subset that is retrieved, leading to some lower scores.

4.2 Robustness Benchmark

To ensure the robustness of my system, I tested against common situations that would result in unintended behavior. Specifically, I evaluated malformed queries and games with very low review counts. To evaluate malformed queries, I tested random keysmashes that would not result in a comprehensible result or question, and questions about information that had no chance of appearing in the retrieved reviews, such as "What is the quality of the PS5 port of this game?" In both of these situations, the system returns "Unable to find answer in retrieved reviews" which is the intended behavior. For games with a low number of reviews, the system performs normally, returning information as long as a relevant review can be found in the retrieved values. For games with 0 reviews, the summary will also not return a hallucinated answer, which is the intended behavior.

5 Future Work

For the possibilities of future additions to this software, the most relevant improvements would be to further engineer the query and summary prompts to be more detailed and minimize extra information. For the retrieval process, preprocessing the query before determining embeddings may improve retrieval results.

A current limitation of the system also lies in its additional review retrieval process. If no answer is found within the current database of review embeddings, the user is given the option to request additional reviews from the Steam API. However, if the model detects an answer that is "good enough" within the supplied context, it will not indicate a request for more reviews. Employing model reranking and accuracy measures to determine when new reviews should be automatically requested would be a beneficial addition.

Ideally, this system would also be adjusted to be flexible in the LLM used, allowing the user to use any API keys they may have access to and customize their own chat interface. This could also lead to further improvements such as storage of user preferences and model fine-tuning to the type of queries users want to search.

References

[1] Florin Cuconasu, Giovanni Trappolini, Federico Siciliano, Simone Filice, Cesare Campagnano, Yoelle Maarek, Nicola Tonellotto, and Fabrizio Silvestri. 2024. The Power of Noise: Redefining Retrieval for RAG Systems. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2024)*. ACM, 719–729. doi:10.1145/3626772.3657834
[2] SteamDB. [n. d.]. https://steamdb.info/stats/releases
[3] SteamReviewSummarizer. [n. d.]. https://steamreviewsummarizer.com/
[4] SteamSummarize. [n. d.]. https://www.steamsummarize.com/
[5] Backlinko Team. 2025. Steam Usage and Catalog Stats. https://backlinko.com/steam-users#how-many-new-games-are-added-to-steam-each-year
[6] yamalight. [n. d.]. https://www.reddit.com/r/Steam/comments/1igylkt/i_built_a_tool_that_summarizes_steam_game_reviews/#lightbox