Steam's games-based search engine

Gustavo Costa
MEIC - FEUP
Porto, Portugal
up202004187@up.pt

João Oliveira
MEIC - FEUP
Porto, Portugal
up202004407@up.pt

Luís Miranda
MEIC - FEUP
Porto, Portugal
up201306340@up.pt

Ricardo Cavalheiro
MEIC - FEUP
Porto, Portugal
up202005103@up.pt

ABSTRACT

In today's day and age, games are becoming increasingly more common and mainstream. With that in mind, we decided that this would be a good topic when it comes to the process of retrieving and preparing information for use in search platforms like Solr[1]. We found a rather large dataset (roughly 81000 entries) in Kaggle[2] containing relevant information, like game names, descriptions, and prices, obtained from Steam[3], one of the biggest and most popular gaming platforms. One of the downsides of having lots of raw information is that it's almost guaranteed that you'll also have a lot of junk or outright useless data so, using the raw datasets as a starting point, we built a Python[4] pipeline capable of cleaning and preprocessing the data, so that it was ready to be used for information searching tasks. In the end, the processed dataset contained 48,219 entries with 10 diverse attributes, encompassing both numerical and textual data types, which roughly translated to half the original size. This was achieved by eliminating uninteresting entries, thereby enhancing the dataset's precision and relevance. For collection, indexing, and data retrieval we use Solr. Out of available query parsers, we chose eDisMax[5] as it is considered the most complete one and can be explored in more depth to produce the most accurate results. In the evaluation process, precision-recall (P-R) curves, precision at 10 (P@10), and average precision metrics are used in conjunction with manual assessment and systematic analysis. Furthermore, we've improved our project by integrating not only semantic search on the queries but also two API's to add aditional layers of information, such as reliable user reviews and visually appealing images. We've also developed a user-friendly web application, which provides a dynamic search page and detailed game listings, ensuring an interactive user experience. The application uses ReactJS[6] for the frontend and NodeJS[7] for the backend, functioning as a RestAPI[8] to facilitate communication with the Solr engine.

CCS CONCEPTS

• **Information systems → Information retrieval**.

KEYWORDS

Data retrieval, Data Processing, Data Transformation, Data Cleaning, Data Integration, Python Pipeline, Steam games

1 INTRODUCTION

Games have become a cultural pillar in today's world of digital and interactive experiences. Given how prevalent games have become in today's society, this paper focuses on them as a thematic research. Steam stands out as an optimal data reservoir owing to its extensive user base and varied game selection, offering a comprehensive dataset that encompasses a diverse spectrum of gaming behaviors. Its digital nature enables precise data collection, encompassing playing statistics, user reviews, and detailed information on an immense quantity of games. This research begins with the identification and initial characterization of the data source, followed by data collection and preparation. The data is then characterized to provide a comprehensive understanding of its details and some information are identified. A collection and indexing of the data was also performed, so that it could be retrieved afterwards. The results were then evaluated based on specific metrics. Finally, we developed a user-friendly web application with a dynamic search page (using semantic search) and detailed game listings, retrieved from two different APIs, to improve the complexity of the information available.

2 DATA SOURCE IDENTIFICATION

The dataset used in this study was obtained from Kaggle[9] (license CC BY-NC-SA 4.0[10]) and consists of two files: `steam_data.csv` (36MB) and `text_content.csv` (148MB), each with about 81,000 entries. While `steam_data` offers a more organized set of information, focusing on non-descriptive features like categories, release dates and developers, `text_content` is rich in textual properties, prominently presenting descriptions and other narrative components relative to each game. The Steam API[11] (free use under ToS[13]) also acts as a data source and is essential for obtaining the initial datasets' missing data, resulting in richer and more complete sets of data that could be used for future analysis.

3 INITIAL DATA CHARACTERIZATION

To be able to preprocess the data, we needed to know what the data looked like so that we could employ the correct procedures. With this in mind, after merging both csvs obtained from Kaggle and removing the duplicated information, we ended up with the following characterization of our data (Table 1).

Attribute	Type	Null Count	Null Percentage
publisher	Text	76190	94.19
pegi	Text	70373	87.00
pegi_url	Text	66098	81.71
desc	Text	31597	39.06
categories	Text	6822	8.43
price	Text	6323	7.82
requirements	Text	6222	7.69
popu_tags	Text	5764	7.13
date	Text	5636	6.97
developer	Text	5547	6.86
all_reviews	Text	5145	6.36
name	Text	5137	6.35
img_url	Text	5134	6.35
user_reviews	Text	5132	6.34
full_desc	Text	1812	2.24
url	Text	0	0.00

Table 1: Attribute types, Null Counts and Percentages for Attributes

- **publisher**: This attribute represents the company that published the game.
- **pegi**: PEGI[14] is a video game content rating system.
- **pegi_url**: URL associated with the PEGI rating.
- **desc**: A brief description of the game.
- **categories**: Represents the genres or categories to which the video game belongs.
- **price**: This attribute contains the cost of the video game.
- **requirements**: This column consists of information about the system requirements or specifications needed to run the video game.
- **popu_tags**: Includes popular tags given for the game by the users and developers.
- **date**: Represents the release date of the game.
- **developer**: Company which developed the game.
- **all_reviews**: Contains game reviews, from users and non-users, statistics.
- **name**: This attribute is the name or title of the video game.
- **img_url**: URL associated with the preview image of the game.
- **user_reviews**: Contains game reviews, from users, statistics. Consists of the following information: overall impression of the review, number of such reviews, and percentage of positive reviews in the last 30 days.
- **full_desc**: Elaborate description of the game. Also contains information like whether it is a game or extra content (i.e. DLC).
- **url**: URL associated with the game. It contains the ID of the game.

At this stage it is important to note a couple of points:

- Typical numerical fields are classified as text fields. In the case of the "price" field, this happens because the numerical price is prefixed with the dollar sign ($).
- The "Null Percentage" column indicates the percentage of nulls per attribute field. For example, 94.19% of the entries in the "publisher" field contain the value "NULL"
- The "url" field contains no "NULL" values because they were automatically removed when the merging of data into a single source occurred, being this field the common field between both files.

4 DATA COLLECTION & PREPARATION

Data preparation and collecting surfaced as challenging when dealing with the initial dataset of our project. The data's initial unstructured state was a big barrier, necessitating careful work to transform disorder into coherence and ensure a solid basis for future analysis and use.

4.1 Pipeline Description

We were tasked with building a pipeline capable of handling and preparing the raw data so that it could be used later. To do so, we chose Python due to its scripting language nature. We also chose to host the data on an SQLite[15] database, instead of the original csv format. SQLite is a fast and simple relational database management system with good Python integration, having many of the needed tools for the job already built in and optimized, allowing for efficient operations.

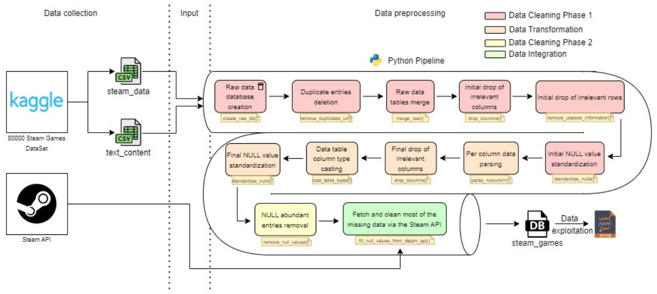
Data collection
Input
Data preprocessing
Data Cleaning Phase 1
Data Transformation
Data Cleaning Phase 2
Data Integration
Python Pipeline
kaggle
80000 Steam Games DataSet
steam_data
text_content
Steam API
Raw data database creation
create_raw_db()
Duplicate entries deletion
remove_duplicates_url()
Raw data tables merge
merge_raw()
Initial drop of irrelevant columns
drop_columns()
Initial drop of irrelevant rows
remove_useless_information()
Initial NULL value standardization
standardize_nulls()
Per column data parsing
parse_<column>()
Final drop of irrelevant columns
drop_columns()
Data table column type casting
cast_table_types()
Final NULL value standardization
standardize_nulls()
NULL abundant entries removal
remove_null_values()
Fetch and clean most of the missing data via the Steam API
fill_null_values_from_steam_api()
DB
steam_games
Data exploitation

Figure 1: Pipeline Diagram

Figure 1 displays the pipeline architecture diagram that our pipeline follows. As observable, the data goes through five distinct phases before it is ready to be used, explained in detail in the following topics.

4.2 Data Collection

Initially, we downloaded a dataset from Kaggle containing information on around 81,000 Steam games. However, the data proved chaotic, demanding extensive cleaning. Many fields ended up with null values, prompting us to consider filling them by making calls to the Steam API. Nevertheless, utilizing these API calls proved to be restrictive due to the rate limits imposed by Steam. Despite allowing 100,000 calls per day, it has a soft limit of approximately 200 calls every 5 minutes, and given the abundance of null values, the process took a considerable amount of time to complete (11 hours).

4.3 Data Preprocessing

Data preprocessing is the phase of cleaning up raw data. This involves deleting duplicates, filling in missing values, and transforming the data into a format that can be understood. It also means figuring out which fields would play an important role in powering the search engine.

4.3.1 *Data Cleaning Phase 1.*

In this phase, the following tasks are performed:

- **Merging Dataset and Removing Duplicates:** Merging datasets and eliminating duplicate records to ensure data integrity and consistency.
- **Dropping Columns:** Removing irrelevant columns (pegi, pegi_url, img_url, popu_tags) to simplify the dataset.
- **Removing Specific Entries:** Removing entries related to DLCs, soundtracks, packs, and new editions, as they have no relevance in this work's context. It also helped to reduce the size of the dataset.
- **Standardizing Nulls:** Standardizing the representation of null values for consistency.

4.3.2 *Data Transformation.*

Data transformation involves reformatting and parsing various data attributes to make them more suitable for analysis. Key tasks include:

- **Parsing Price Attribute:** Standardizing price formats for numerical analysis.
- **Parse Full Description:** Removed 'About this' prefix for all the entries for better readability.
- **Parsing Categories:** Parsing and standardizing game categories for better categorization.
- **Parsing Date:** Formatting date values consistently (e.g., DD MM, YYYY).
- **Parsing System Requirements:** Breaking down complex system requirement data into structured components.
- **Parsing Reviews:** Extracting and parsing the percentage of positive reviews from the textual review information.
- **Parsing Developer:** Standardizing developer names for easier analysis.
- **Dropping Columns (user_reviews, publisher):** Removing columns that are not relevant to the analysis. The publisher contained a high percentage of null values.

4.3.3 *Data Cleaning Phase 2.*

The need for a second phase of data cleaning arises from inconsistencies introduced during the parsing and transformation steps:

- **Standardizing Nulls:** Continuing the standardization of null values to ensure data consistency.
- **Removing Entries with more than 3 Null Values:** Removing records with more than 3 missing values while considering the impact on data integrity.

4.3.4 *Data Integration.*

Data integration is the final phase, focusing on filling in missing data using external sources, specifically the Steam API.

- **Filling Null Values with Data from the Steam API:** Retrieving missing data from the Steam API to enhance the dataset's completeness and accuracy.

4.4 Conceptual Data Modelling

After preprocessing a clear model for our data can be defined, as seen in Figure 2.

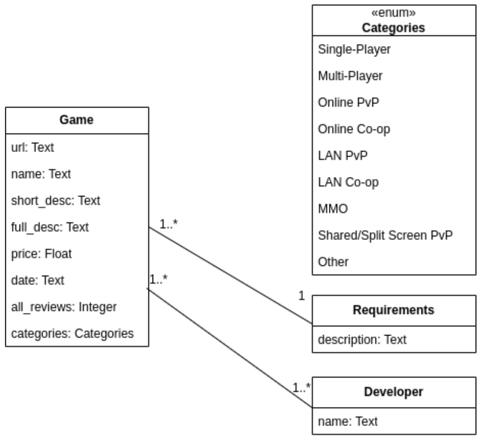

Figure 2: Conceptual Data Model

Only three entities are identifiable, being the game, its requirements, and its developer. Other than that, a game can have a list of categories that easily help with identifying if a certain game is desired when a certain information need is carried out.

5 DATA CHARACTERIZATION

After running the pipeline through the initial dataset, the processed data ended up with exactly 48219 entries and 10 features, with both numerical and textually rich attributes.

5.1 Collection Characterization

Attribute	Type	Null Values	Null Percentage (%)
all_reviews	Integer	6532	13.54
price	Float	677	1.40
developer	Text	82	0.17
desc	Text	65	0.13
requirements	Text	42	0.09
date	Text	41	0.09
full_desc	Text	29	0.06
url	Text	0	0.00
categories	Text	0	0.00
name	Text	0	0.00

Table 2: Final Dataset Attributes, Type, Null Count, and Null Percentage

Compared to the original dataset, the refined version now boasts approximately 48,000 entries—roughly half of its initial size. This reduction is primarily attributed to the elimination of redundant entries, such as DLCs and soundtracks, streamlining the dataset for greater precision and relevance.

5.2 Free & Paid Game Releases

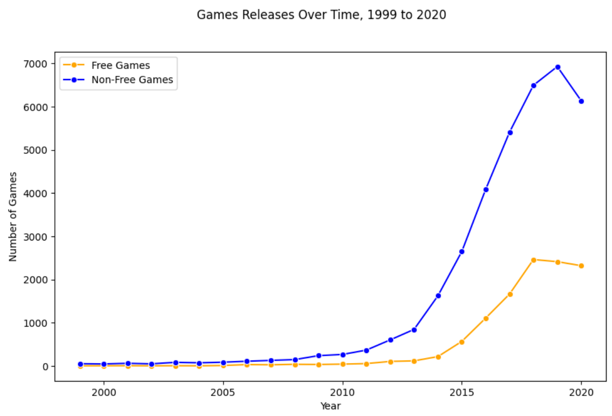

Figure 3: Games Released from 1999 to 2020

Figure 3 illustrates an increase in game releases over time, with one standout trend: most of these games are not for free. This indicates a shift in developer preferences toward paid models, which may be caused by reasons like rising development costs or a desire for higher financial rewards. The amount of games available is growing and so are the monetization methods used.

5.3 Game Review Evolution

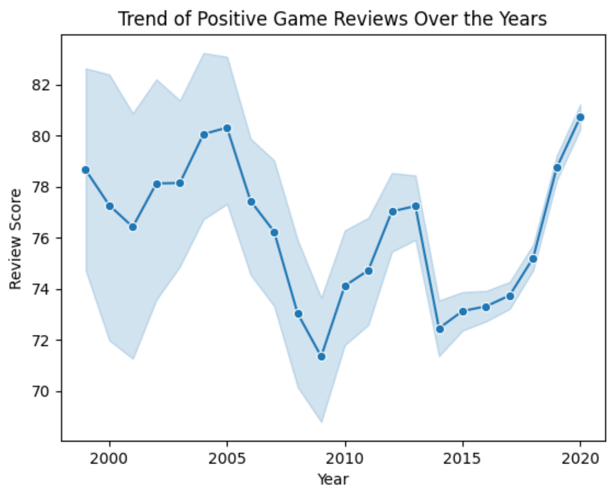

Figure 4: Average Positive Game Reviews from 1999 to 2020

It's clear from Figure 4 that, despite the oscillations, the overall trend seems to be that the users are enjoying the different games. It's interesting how the plot showcases a consistent range of 70-80% positive reviews per year.

5.4 Text Analysis

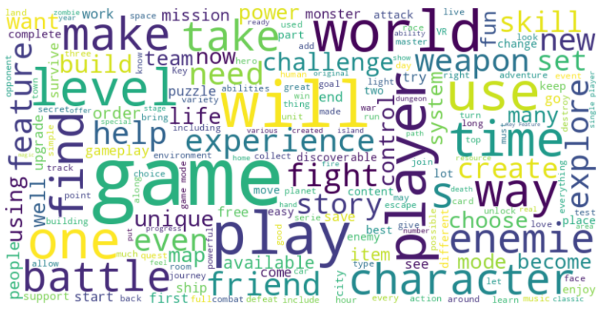

Figure 5: Word Cloud Image

We made a wordcloud (Figure 5) based on the attribute 'full_desc', which is the one to be most likely used in the search engine as it is the richest in textual data. We can observe that common words like 'Game', 'Play', 'Battle', and 'Fight' are predominant, which goes according to our expectations.

6 INFORMATION NEEDS

- Single-player games released in the last decade with an underwater setting
- Historical games where the player can command an army
- 2D platform game where the main character is an animal
- Highest-rated open-world RPGs set in fantasy realms

7 COLLECTION & INDEXING

Now with a near-ready-to-use dataset, it's time to make the last transformations to the data so that it's ready to be uploaded to and indexed by Solr, the suggested tool.

7.1 Document Definition

In our system, a game translates 1:1 to a document. As there will be only one sort of document, we will only be querying one collection. Initially, our dataset was stored in an SQLite database, but due to Solr's superior JSON support, we converted our data to this format. Although Solr has a default schema, we found it more advantageous to create a custom schema that meets our needs. Before actually populating the collection, some parsing was necessary to ensure consistency in the chosen field types for each document property.

- As SQLite lacks support for lists, certain attributes were represented as strings with separators for individual elements. In the process of converting to JSON, we transformed them into a list of strings.
- The price was parsed according to a particular format so that a CurrencyFieldType field could be included in our schema.
- The date was also converted to ISO-8601 format so that it would match Solr's built-in date field type.

7.2 Schema Details

Keeping our information needs in mind, we designed a schema that would allow for efficient querying. Table 3 shows the fields for each document that we defined in the schema. Note that the field types displayed in italics are the custom field types we made, that better fit in with our documents.

Name	Type	Indexed
url	string	False
name	*text_short*	True
date	*date_field*	True
categories	*text_list*	True
all_reviews	pint	True
developer	*text_short*	True
desc	*text_long*	True
full_desc	*text_long*	True
price	*price_field*	True
*_l_ns	plong	True
*_s_ns	string	True

Table 3: Document schema fields

Since each field type needs a tokenizer and a filter for indexing and querying, we also had to define those, where the default Solr ones just wouldn't cut it. The primary examples of this are the main document search targets: field desc and full_desc. As with many natural language processing (NLP) tasks, some actions need to be taken to enhance the quality and performance of searching for documents. For this, Solr has built-in features that allow for tokenization, ASCII folding, ignore casing, stemming, stop word filtering, and synonym graph filtering, which we took advantage of. In Table 4 and Table 5, the tokenizer and filters we chose for either indexing the documents or querying the collection, are available for further information on what each custom-defined field does. Note that the default Solr values are denoted in italics, which means that this field either manipulated the data inside it in some other way or was just a symbolic and more meaningful way to represent the data type.

Name	Index Tokenizer	Index Filters
text_list	StandardTokenizerFactory	ASCIIFoldingFilterFactory LowerCaseFilterFactory
date_field	*Default*	*Default*
price_field	*Default*	*Default*
text_short	StandardTokenizerFactory	ASCIIFoldingFilterFactory LowerCaseFilterFactory ClassicFilterFactory KStemFilterFactory
text_long	ClassicTokenizerFactory	ASCIIFoldingFilterFactory LowerCaseFilterFactory StopFilterFactory ClassicFilterFactory KStemFilterFactory SynonymGraphFilterFactory

Table 4: Custom defined field types' index analyzer definition

Name	Query Tokenizer	Query Filters
text_list	*Default*	*Default*
date_field	*Default*	*Default*
price_field	*Default*	*Default*
text_short	StandardTokenizerFactory	ASCIIFoldingFilterFactory LowerCaseFilterFactory ClassicFilterFactory KStemFilterFactory
text_long	ClassicTokenizerFactory	ASCIIFoldingFilterFactory LowerCaseFilterFactory StopFilterFactory ClassicFilterFactory KStemFilterFactory

Table 5: Custom defined field types' query analyzer definitions

The index and query filters are designed to optimize the retrieval process. Inputs start by being tokenized then we ASCII fold them, which converts all non-ASCII characters into their ASCII equivalent. Afterward, all tokens are converted into lowercase and, in the case of fields of type text_long like the game description, all the stop words are removed. Once that's done we strip all the periods from acronyms and "'s" from possessives using Solr's ClassicFilterFactory and then we stem the tokens. At indexing time and for text_long fields, a synonym graph is also built to improve the matching of documents to queries.

7.3 Indexing Process

We initiated the process by establishing a Docker[16] container featuring a Solr v9.3.0 image, within which we generated a core named "games." Subsequently, we created a script responsible for transferring essential files required for the schema to function like the synonyms and currency files, and uploading the schema, followed by the documents. This was all achieved using either Solr's HTTP API[17] or Docker commands.

8 DATA RETRIEVAL

Similar to how relational database managing systems, like SQLite, the one we used, allow for querying of the database, Solr allows for querying of a collection. This approach allows for precise adjustments to the majority of the query's parameters.

8.1 Queries

To fulfill the information needs presented earlier, out of the query parsers available in Solr we opted for the eDisMax query parser, as it offers the most extensive set of functionalities. For each information need we constructed two queries, a basic one where we don't explore much of the features eDisMax offers us and a more complex one where we got more creative and explored options like wildcards, fuziness, proximity, and term boosting.

8.2 Single-player games released in the last decade with an underwater setting

This query aims to retrieve single-player games where the gameplay takes place underwater. To guarantee that all the matches were developed within the last decade we used a filter query with this specific constraint. To filter games where both words existed but were unrelated, we investigated the proximity between water/underwater and the setting in the enhanced query. Since "underwater" was the primary focus of the information need, we also gave it a higher boost. However, we preserved the term "water" because it was occasionally used in games that satisfied this information need.

Basic Query: ("water setting"~3) or ("underwater setting"~2)

- **Query Operator (q.op):** AND
- **Filter Query (fq):** categories: single-player date:[NOW/DAY-3653DAYS TO NOW]
- **Query Fields (qf):** full_desc desc
- **Basic Boost (bq):** None
- **Boost Functions (bf):** None

Enhanced Query: ("water setting"~3)^2 or ("underwater setting"~2)^4

- **Query Operator (q.op):** AND
- **Filter Query (fq):** categories: single-player date:[NOW/DAY-3653DAYS TO NOW]
- **Query Fields (qf):** full_desc desc
- **Basic Boost (bq):** None
- **Boost Functions (bf):** None

8.3 Historical games where the player can command an army

The goal of this search is to locate historical games in which the player can lead an army. The only real distinction between the two queries is the weights assigned to "historical" and "command army". We included a linear boost function that raises the score in proportion to the number of reviews for the game in an attempt to ensure that the most "popular" games (with a higher score) were retrieved first.

Basic Query: (historical) and (command army)

- **Query Operator (q.op):** AND
- **Filter Query (fq):** None
- **Query Fields (qf):** full_desc desc
- **Basic Boost (bq):** None
- **Boost Functions (bf):** None

Enhanced Query: (historical)^2 and (command army)^3

- **Query Operator (q.op):** AND
- **Filter Query (fq):** None
- **Query Fields (qf):** full_desc desc
- **Basic Boost (bq):** None
- **Boost Functions (bf):** linear(all_reviews,0,100)^1.5

8.4 2D Platform Games where the main character is an animal

Finding 2D platform games with animals as the primary characters is the goal of this query. We apply a wildcard on "platform" to allow it to find documents containing similar terms (for example "platformer" or "platformed"). On the basic query, we try to match documents that have either "play" or "control" and animal. To make the terms in the improved query more precise and near to one another, we also included proximity to avoid unwanted results.

Basic Query: "2D platform*"~2 ((play animal) or (control animal))

- **Query Operator (q.op):** AND
- **Filter Query (fq):** None
- **Query Fields (qf):** full_desc desc
- **Boost Query (bq):** None
- **Boost Functions (bf):** None

Enhanced Query: "2D platform*"~2 ("play as animal"~3^2 or "play with animal"~3^2 or "control animal"~3^2)

- **Query Operator (q.op):** AND
- **Filter Query (fq):** None
- **Query Fields (qf):** full_desc desc
- **Boost Query (bq):** None
- **Boost Functions (bf):** None

8.5 Highest-rated open-world RPGs set in fantasy realms

This query aims to retrieve open-world RPGs set in fantasy realms with the best ratings. Other than applying a boost to the enhanced, both queries are rather identical. If the word "RPG" occurs in the game's title, this boost mechanism takes that into account and also increases the score exponentially based on user reviews. We thought that employing this boost function would be preferable to just sorting the games based on reviews, as doing so would ignore each document's score in the retrieval order.

Basic Query: "open world"~2 "RPG" "fantasy realms"~2

- **Query Operator (q.op):** AND
- **Filter Query (fq):** None
- **Query Fields (qf):** full_desc desc
- **Boost Query (bq):** None
- **Boost Functions (bf):** product(if(exists (query(!v='name:"RPG"')),2,1), exp(product(all_reviews,0.03)))

Enhanced Query: "open world"~2^2 "RPG"^3 "fantasy realms"~2^2

- **Query Operator (q.op):** AND
- **Filter Query (fq):** None
- **Query Fields (qf):** full_desc desc
- **Boost Query (bq):** None
- **Boost Functions (bf):** product(if(exists (query(!v='name:"RPG"')),2,1), exp(product(all_reviews,0.03)))

9 EVALUATION

To successfully compare our different approaches, we set up three different environments which we called "Systems".

- **System 1** - Solr's schemaless mode, which we tested with the basic queries;
- **System 2** - Our custom-made schema, which we also tested with the basic queries;
- **System 3** - Similar to System 2, but we tested the enhanced queries.

By doing so we can determine the impact of the schema, by comparing System 1 with System 2, and also analyze the efficiency between both versions of the query, by comparing System 2 with System 3.

As a way to draw meaningful comparisons between both schemas and query versions, we must evaluate the outcomes methodically.

Evaluating queries is tough since different users will determine whether documents are relevant in different ways. It is nearly impossible to manually analyze the full dataset in order to identify relevant documents. Consequently, our methodology for addressing each information need involved employing a Python script to iterate through potentially relevant games (containing keywords related to the information need) and manually marking those deemed pertinent.

To effectively compare the performance of both queries, we will employ the following metrics:

- **Precision at N (P@N)** - Precision quantifies the proportion of pertinent documents recovered relative to the total number of documents retrieved (N).
- **Recall** - Recall is a measure of the completeness of a search or information retrieval system. It is calculated as the number of relevant documents retrieved divided by the total number of relevant documents in the collection. Once it's practically impossible to find all the relevant documents, we calculated recall the following way: the relevant documents retrieved divided by the sub set of documents we found relevant. (qrels)
- **Average Precision (AP)** - AP is defined as the average of the precision values obtained for the set of top N documents existing each time a new relevant document is retrieved.
- **Precision-Recall Curve** - The precision-recall curve shows the tradeoff between precision and recall for different thresholds. The area under the curve functions as a quality indicator, and it shows the trade-offs between these measures at different levels. Greater overall precision is indicated by a wider area.

We opted for **N=10** since it was approximately the number of items a Google[12] Search Results page used to have. This choice is underscored by the understanding that users, in practice, may be less inclined to delve deeper into search results beyond this threshold.

9.1 Single-player games released in the last decade with an underwater setting

Metric	System 1	System 2	System 3
Average Precision	0.698231	0.771713	0.851151
Precision at 10 (P@10)	0.700000	0.800000	0.900000
Individual Assessment	NNRRRRRRNR	NRRRRRRRNR	RRRRRRRNRR

Table 6: 1st Query Evaluation Metrics

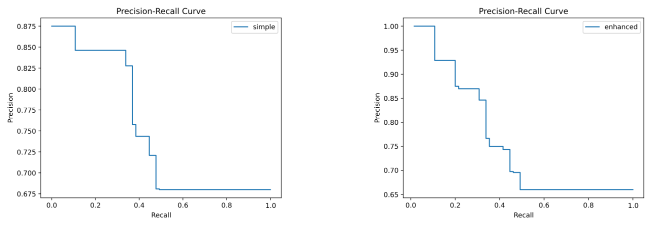

Figure 6: 1st Query Performance Comparison

9.2 Historical games where the player can command an army

Metric	System 1	System 2	System 3
Average Precision	0.416588	0.695610	0.725188
Precision at 10 (P@10)	0.300000	0.700000	0.800000
Individual Assessment	RNRNNNNNRN	RRNRRNRRRN	RRRRNRRNRR

Table 7: 2nd Query Evaluation Metrics

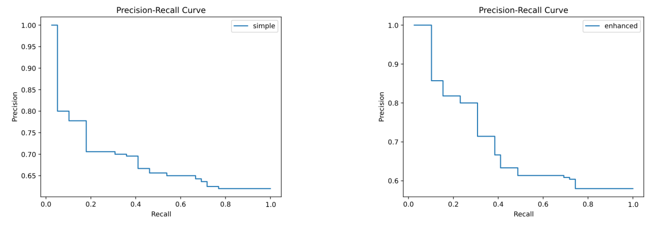

Figure 7: 2nd Query Performance Comparison

9.3 2D platform game where the main character is an animal

Metric	System 1	System 2	System 3
Average Precision	0.807787	0.727764	0.864219
Precision at 10 (P@10)	0.700000	0.800000	0.800000
Individual Assessment	RRNRRRNRRN	RRRNRRRRRN	RRRRRRNRRR

Table 8: 3rd Query Evaluation Metrics

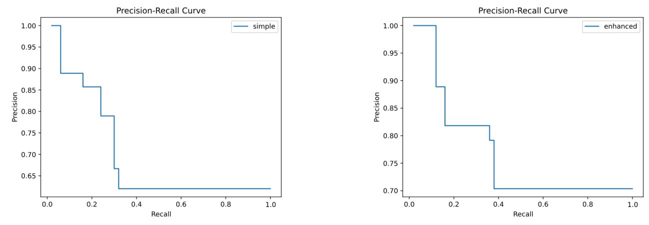

Figure 8: 3rd Query Performance Comparison

9.4 Highest-rated open-world RPGs set in fantasy realms

Metric	System 1	System 2	System 3
Average Precision	0.000000	0.873773	0.885226
Precision at 10 (P@10)	0.000000	0.900000	1.000000
Individual Assessment	NNNNNNNNNN	RRRRRRRRRN	RRRRRRRRRR

Table 9: 4th Query Evaluation Metrics

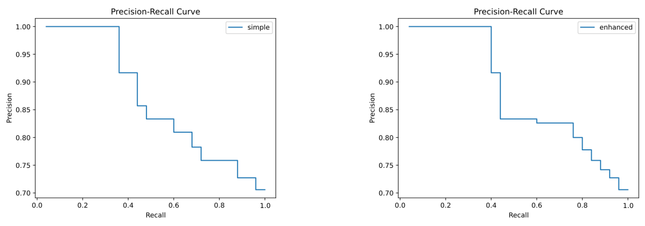

Figure 9: 4th Query Performance Comparison

9.5 Evaluation Interpretation

It is possible to observe that the systems generally perform as anticipated, with System 1 exhibiting the lowest performance and System 3 demonstrating the highest. The disparity in results between System 1 and 2, though not as substantial as initially assumed, primarily arises from the schemaless mode in which Solr employs minimal to no filters or analyzers. In contrast, our predefined schema incorporates essential steps crucial for the engine to identify desired results. This discrepancy may stem from the support of synonyms in our schema, theoretically enhancing result diversity but potentially introducing redundancy by matching non-relevant documents and also the removal of stopwords. On Table 8 despite System 2 having a higher P@10 when compared to System 1, the latter has a higher Average Precision. This may happen because the system manages to retrieve relevant items later in the ranking. An intriguing observation is that System 1 fails to retrieve any document for the latest information need, as seen in Table 9, possibly attributed to the absence of synonyms in the schema. System 3 consistently demonstrates superior performance in comparison to System 2, thanks to the implementation of boosts on the most relevant query terms and the utilization of Solr features that are not employed in the more basic query.

Metric	System 1	System 2	System 3
Mean Average Precision	0.480652	0.767215	0.831446

Table 10: Global Evaluation Metrics

10 SEARCH SYSTEM

When combining both the boosted queries and the improved schema we obtained significantly better results when compared to the baseline system. However, as we input specific Solr settings (boosts) for each query, these results may be deceptive if the search engine is utilized more frequently. The third task's objective is to investigate strategies to enhance the search system overall. In light of this, we will assess various system performances against a new hypothesis, which is semantic search, by utilizing the Mean Average Precision (MAP) as the primary assessment metric. To enhance the authenticity of our search engine experience, we've integrated a user interface through a web application. This enhances user-friendliness and offers a setting for users to interact with the search engine.

10.1 Information Expansion

Now we are utilizing three additional sources of information to improve the complexity of every game listing on our platform: Steam game reviews[18], Game posters[19], and Game trailers[20]. We've added extra layers of information to enhance each document, in addition to the essentials like title, price, release date, and description. Through the extraction of user evaluations from reliable sources, we have given players insightful information about the opinions of the gaming community regarding each game. Additionally, we've added images to the listings to enhance the whole experience's visual appeal in addition to its informational value. This deliberate effort guarantees that users engage with our program in a more comprehensive and pleasurable way while also giving the material more depth.

10.2 User Interface

To enhance the user experience we implemented a web application using ReactJS for the frontend and NodeJS for the backend. The backend, functions as a RestAPI, facilitating the communication with the Solr engine, allowing us to retrieve documents tailored to the user's preferences. The application unfolds across three thoughtfully designed pages. The initial home page not only sets the stage with background information but also conveniently presents users with popular and straightforward query—making. The search page, designed with user freedom in mind, offers a dynamic experience where users can delve into personalized searches, aided by intuitive filters for added convenience. Lastly, we've dedicated a page exclusively to each document, ensuring that users can delve into a comprehensive display of information, providing a smooth experience.

10.3 Semantic Search

As previously stated, we've attempted to implement a mechanism for semantic search. Initially, this sounded like a good idea, given that we had some difficulties balancing the meaning of some context-dependent words. To do this, we used the Sentence Transformers Python library[21], which allowed us to pass every single document in our database through a model called "all-MiniLM-L6-v2"[22] that "maps sentences paragraphs to a 384-dimensional dense vector space and can be used for tasks like clustering or semantic search". Having said dense vector space (or embedding) for each document, we could then do the same procedure for each query we want to run against our collection and, using the k-nearest neighbors algorithm[23], "compare" vector spaces. On a side note, since we had to run a model for every single document, and some documents can hold quite a lot of information, the standard approach of iterating sequentially through the documents was a no-go. That meant we had to implement optimizations of our own to be able to have our whole dataset embedded. We went through two optimization steps that drastically improved performance: CPU parallelization and GPU model running. We will not be going into depth as to how we did those.

11 EVALUATION

To evaluate the impact of semantic search, we used the same metrics as previously to compare the different systems. We called this new system "System 4" and we compared its results to the ones we obtained by System 3, the one with the enhanced query and schema, and saw if the results improved with the changes. Initially, we used the edismax parser for both systems, but we quickly realized that the new system had terrible performance. More specifically, the new system with the edismax parser had a P@10 of 0 for all queries tested. We tried, unsuccessfully, to find an answer as to why this could be happening. We thought that maybe the synonym list was affecting the way the parser perceived the semantics of a query, but that wasn't the case. Our best guess is that, somehow, the flexibility of the parser doesn't go well with the broadening effect of semantic search. On the other hand, we did try swapping to the Lucene parser which seemed to improve performance, when compared to System 3. The problem with the Lucene parser is that it's much less flexible, so we took a different approach: we simplified our queries to contain only natural language. This also has the advantage of making our otherwise complex and unnatural queries to be similar to a user's natural input.

11.1 Single-player games released in the last decade with an underwater setting

Query: underwater setting game

- **Query Operator (q.op):** AND
- **Filter Query (fq):** categories: single-player date:[NOW/DAY-3653DAYS TO NOW]

Metric	System 3	System 4
Average Precision	0.851151	0.773248
Precision at 10 (P@10)	0.900000	1.000000
Individual Assessment	RRRRRRRNRR	RRRRRRRRRR

Table 11: 1st Query Evaluation Metrics

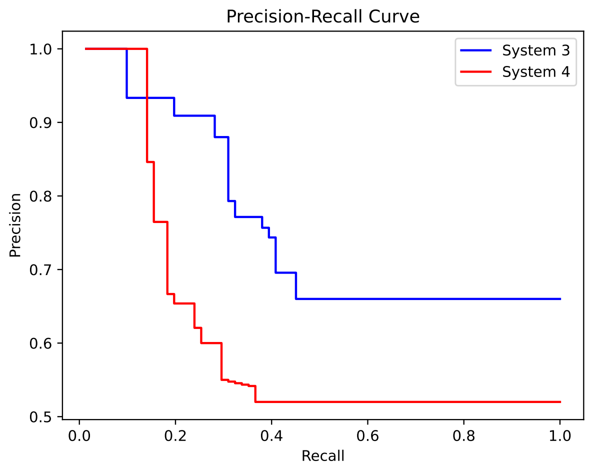

Figure 10: 1st Query Performance Comparison

11.2 Historical games where the player can command an army

Query: historical game where you command an army

- **Query Operator (q.op):** AND
- **Filter Query (fq):** None

Metric	System 3	System 4
Average Precision	0.725188	0.739764
Precision at 10 (P@10)	0.800000	0.800000
Individual Assessment	RRRRNRRNRR	RRRRRRRNNR

Table 12: 2nd Query Evaluation Metrics

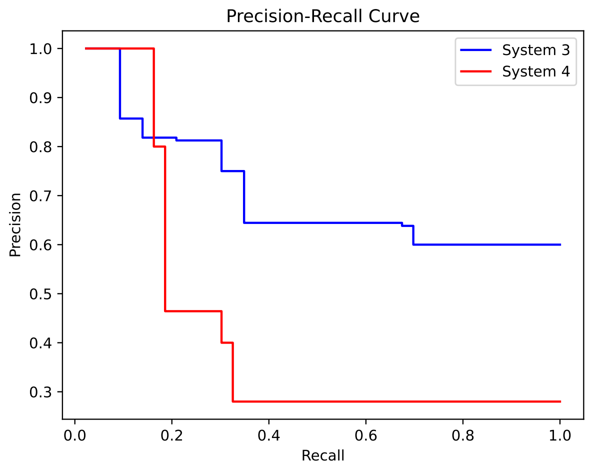

Figure 11: 2nd Query Performance Comparison

11.3 2D Platform Games where the main character is an animal

Query: platform game where you play as an animal

- **Query Operator (q.op):** AND
- **Filter Query (fq):** None

Metric	System 3	System 4
Average Precision	0.864219	0.309524
Precision at 10 (P@10)	0.800000	0.200000
Individual Assessment	RRRRRRNRRN	NNRNNNRNNN

Table 13: 3rd Query Evaluation Metrics

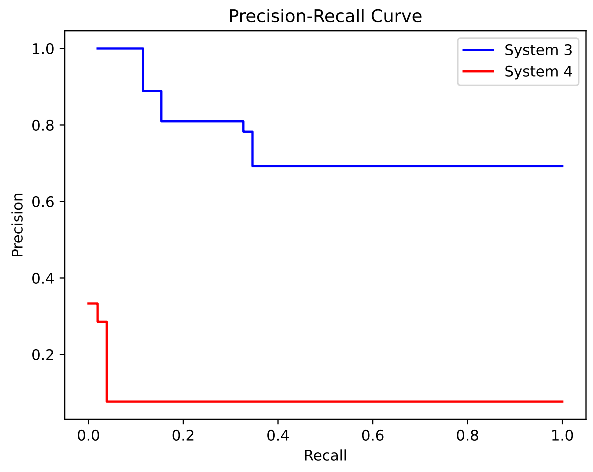

Figure 12: 3rd Query Performance Comparison

11.4 Highest-rated open-world RPGs set in fantasy realms

Query: open world RPG with fantasy realms

- **Query Operator (q.op):** AND
- **Filter Query (fq):** None

Metric	System 3	System 4
Average Precision	0.885226	0.988889
Precision at 10 (P@10)	1.000000	0.900000
Individual Assessment	RRRRRRRRRR	RRRRRRRRNR

Table 14: 4th Query Evaluation Metrics

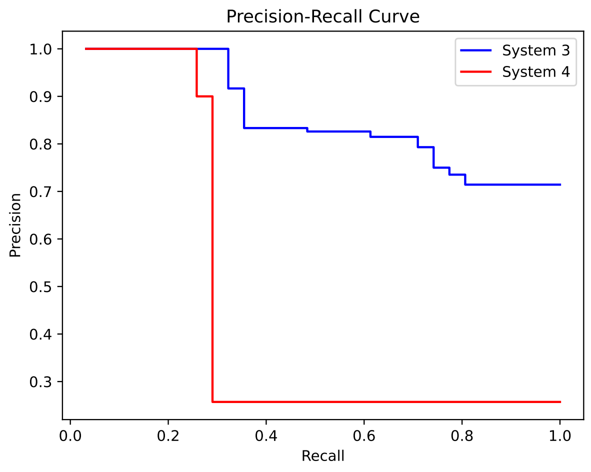

Figure 13: 4th Query Performance Comparison

11.5 Evaluation Interpretation

In the context of the improvements, the system enhancements exhibited some performance disparities. For the first query, while we obtained a P@10 of 1.0, we obtained a lower average precision meaning that, after the 10 first results, fewer relevant documents were obtained. The performances for the second query were very similar, obtaining the same P@10 and approximately the same average precision. On the third query System 4 was completely outperformed by the previous system, obtaining a very low value at both metrics. This could be a result of the parser not understanding the context of the query itself, as these queries were formed using "normal language" so to speak. For the last query the system with the improvements performed better in the long term than System 3 as the average precision is higher, however found less relevant documents in the first 10 documents. System 4 exhibits similar results to System 3, demonstrating occasional superiority and inferiority without significant oscillations, except notably in the third query. The anomaly in performance during the third query suggests potential challenges in contextual understanding by the parser and is what causes the discrepancy in the Mean Average Precision.

Metric	System 3	System 4
Mean Average Precision	0.831446	0.7028656

Table 15: Global Evaluation Metrics

12 CONCLUSIONS & FUTURE WORK

After rigorous data collection and indexing, we have successfully transformed an unstructured dataset into a well-structured and comprehensive collection. This dataset, enriched with both textual and numerical elements, was then ready for future use. In the following stages, we defined a document structure, converted our data into JSON for Solr compatibility, and created a custom schema for efficient querying. Our indexing process was automated using Docker and Solr. Queries were crafted using the eDisMax query parser in Solr to fulfill specific information needs. Afterward, we evaluated our system's performance using metrics such as Precision-Recall, P@10, and AP. The results provided valuable insights into the effectiveness of our retrieval strategies and algorithms and despite not being as discrepant as we thought, they still followed our initial expectations of System 1 (Solr's schemaless mode) being the worst one and System 3 (our custom-made schema, with enhanced queries) the best performance-wise. We improved our search system's user experience by implementing a user-friendly interface, making it more intuitive to the user, as building a query is a lot more straight-forwarded. We also implemented semantic search, however it didn't turn out as we expected since the results were similar, if not worse, to what we accomplished with System 3. Last but not least, despite the initial challenges encountered and the unstructured nature of the original dataset, our efforts have culminated in a robust and efficient system capable of handling complex queries and delivering accurate results in a seamless experience for the user.

REFERENCES

[1] Solr. *https://solr.apache.org/* Last accessed on 16/11/2023.
[2] Kaggle. *https://www.kaggle.com/* Last accessed on 16/11/2023.
[3] Steam. *https://store.steampowered.com/* Last accessed on 16/11/2023.
[4] Python. *https://www.python.org/* Last accessed on 16/11/2023.
[5] eDisMax Query Parser *https://solr.apache.org/guide/solr/9_3/query-guide/edismax-query-parser.html* Last accessed on 16/11/2023.
[6] React - A JavaScript library for building user interfaces *https://reactjs.org/* Last accessed on 14/12/2023.
[7] Node.js - JavaScript runtime built on Chrome's V8 JavaScript engine *https://nodejs.org/* Last accessed on 14/12/2023.
[8] Representational state transfer (REST) *https://www.restapitutorial.com/* Last accessed on 14/12/2023.
[9] Dataset on Kaggle. *https://www.kaggle.com/datasets/deepann/80000-steam-games-dataset* Last accessed on 16/11/2023.
[10] CreativeCommons. *https://creativecommons.org/licenses/by-nc-sa/4.0/* Last accessed on 16/11/2023.
[11] SteamDev. *https://steamcommunity.com/dev* Last accessed on 16/11/2023.
[12] Google. *https://www.google.com/* Last accessed on 16/11/2023.
[13] Steam's ToS. *https://steamcommunity.com/dev/apiterms* Last accessed on 16/11/2023.
[14] PEGI. *https://pegi.info/* Last accessed on 16/11/2023.
[15] SQLite. *https://www.sqlite.org/index.html* Last accessed on 16/11/2023.
[16] Docker *https://www.docker.com* Last accessed on 16/11/2023.
[17] Solr's HTTP API Docs *https://solr.apache.org/guide/solr/9_3/index.html* Last accessed on 16/11/2023.
[18] Steam Game Reviews *https://store.steampowered.com/appreviews* Last accessed on 14/12/2023.
[19] Game Poster Images *https://serpapi.com/search* Last accessed on 14/12/2023.
[20] Game Trailers *https://store.steampowered.com/app* Last accessed on 14/12/2023.
[21] Sentence Transformers Library *https://huggingface.co/sentence-transformers* Last accessed on 14/12/2023.
[22] all-MiniLM-L6-v2 *https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2* Last accessed on 14/12/2023.
[23] k-nearest neighbors algorithm *https://en.wikipedia.org/wiki/K-nearest*$_{n}eighbors_{a}lgorithmLastaccessedon$ 14/12/2023.