Design and Implementation of a Semantic Search Engine for Anime Metadata: A Case Study on One Piece using Apache Solr

João Lourenço
up202108863@up.pt
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal

Tiago Cruz
up202108810@up.pt
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal

Tomás Xavier
up202108759@up.pt
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal

João Cardoso
up202108732@up.pt
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal

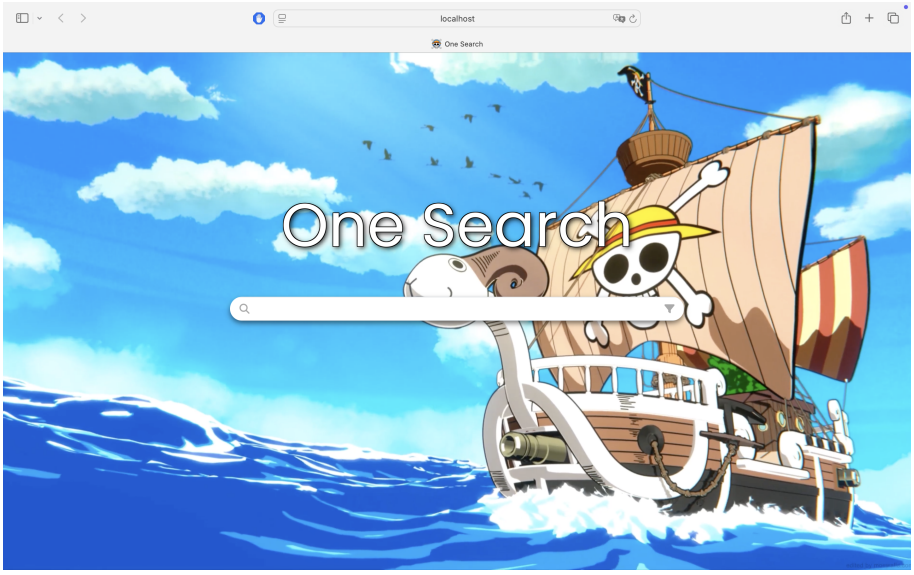

Figure 1: Indexing & Retrieval process diagram using Apache Solr

Abstract

This project presents a data processing pipeline for scraping and analyzing information from the One Piece fandom wiki, specifically targeting episode details of the One Piece anime, in order to solve the problem of tediously searching for information regarding one of the longest-running anime of all time. Using the Beautiful Soup library in Python, the project retrieves key information, including episode titles, air dates, and summaries. The extracted data is cleaned and structured into a usable format for further analysis.

The dataset contains all 1120 episodes aired until the day of writing this, the 8th of October of 2024. This information was indexed into **Apache Solr** in order to develop a search engine targeting One Piece episodes in specific. A user interface was developed as a website, built using **React** [8] for the **frontend** and **Django** [15] for the **backend**. A query made by the user is sent from the frontend to the backend, which routes it over to **Solr**. The response from **Solr** is processed by the **backend** and, finally, returned to the **frontend** for user visualization. The website provides a search bar, where the user can write a query to take advantage of the search system. Additionally, a list of filters are provided so the user can narrow down the results e.g, the user wants to search for "*zoro pirate hunter*", but only wants results from certain Arcs and Sagas of the anime. Finally, the user can obtain information regarding an episode without using the search engine at all, by appending "episode/<episode_id>" to the URL (<episode_id> being an integer representing the episode number).

G62, October 13–10, 2024, Porto, Portugal

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

Keywords

One Piece, Data, Scraping, Fandom, Anime

ACM Reference Format:
João Lourenço, Tiago Cruz, Tomás Xavier, and João Cardoso. 2024. Design and Implementation of a Semantic Search Engine for Anime Metadata: A Case Study on One Piece using Apache Solr. In *Proceedings of (G62).* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

One Piece is one of the longest-running anime of all time with over 1000 episodes, by the time of writing this article. With so many episodes and such an extensive world, it can prove difficult to search for when a specific event happened in the anime. Therefore, this project addresses the need for an automated and efficient way to retrieve detailed information about the One Piece anime. By scraping data from the Fandom Wiki using the **Beautiful Soup** library in **Python**, we collect and organize information about each episode in a structured format. The ultimate goal of this project is to build a search engine powered by **Apache Solr**, enabling users to quickly and easily search for specific details regarding One Piece episodes. This tool will not only provide rapid access to episode data but also allow for advanced queries, facilitating the experience of searching for a small piece of information about such an extensive world.

2 Data Sources

2.1 Identification

The primary data source for this project is the One Piece Fandom Wiki[6], a comprehensive fan-driven wiki dedicated to the One Piece anime and manga series. The wiki provides detailed information about each episode, including titles, air dates, plot summaries, and related metadata. It is frequently updated by fans, ensuring that the latest episodes and information are available shortly after release. The secondary source of data comes from subtitle websites such as subdl[5] and opensubtitles[13], which have downloadable files that contain the subtitles of the episodes in english.

2.2 Characteristics

Most of the data is available in an **HTML** format on the One Piece wiki [6]. The scraped information includes various text fields such as episode, titles, air dates, opening name, ending name, anime notes and description referred to as *summary*. For the subtitles, after downloading the subtitle files from subdl[5] and opensubtitles [13], the files can then be parsed in order to obtain the episode's script or episode's subtitles. Post-scraping, the data is structured into a CSV file for later processing.

The One Piece anime has, at the time of writing this, 1120 episodes. This results in a dataset that includes 1120 rows of structured data, with each row containing multiple fields (Episode, Title, Season, Arc, Saga, Air Date, Opening, Ending, Summary, Episode Script).

The One Piece Fandom Wiki[6] content is licensed under the **Creative Commons Attribution-Share Alike License 3.0 (Unported) (CC BY-SA)**. This license states that all users editing or otherwise contributing to wikis that use the **CC BY-SA** license agree to grant broad permissions to the general public to re-distribute and re-use their contributions freely for any purpose, including commercial use, in accordance with the **CC BY-SA** license. Such use is allowed where attribution is given and the same freedom to re-use and re-distribute applies to any derivative works of the contributions.

Regarding the subtitles, opensubtitles claims that the files they offer are **not** illegal warez downloads[14]. Furthermore, they also state that these documents may not be freely distributed, but may be used and distributed for non-commercial, scientific and educational purposes. In conclusion, we have permission to use these subtitles files for this project, seeing as it is non-commercial.

3 Collection & Preparation

3.1 Pipeline Description

The extraction of information relevant for this task can be made using a data pipeline. In the context of this task, the data pipeline consists in the process of producing a scraper using **Python's Beautiful Soup** library[17] to retrieve the information from the One Piece Fandom wiki[6] and after the initial step of downloading the subtitles from subdl[5] and opensubtitles[13], a simple python program can parse the subtitles one by one and feed them into the cleaning and preparation of data segment.

3.2 Pipeline Diagram

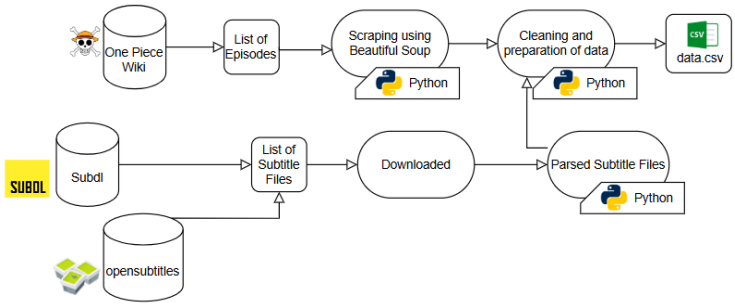

Figure 2: Pipeline Data Diagram

Figure 2 indicates the data diagram for the pipeline process. The information comes from the One Piece Fandom wiki website[6] which contains a webpage for each episode of the anime. Regarding the subtitles, the information comes from subdl[5] and opensubtitles[13]. Then our scraper is used to retrieve the desired information. Finally, this information is properly cleaned and structured into a CSV file for further exploratory analysis.

3.3 Collection operations

The process of collecting data requires to first analyze the fields of data that are present in the anime's wiki pages that are of use to this task. The wiki has an episode guide page that provides links to each episode's page, organized by arc in chronological order.

After entering the episode page the information that is useful for the process is the following: **Arc**, **Saga**, **Episode Number** and

Title, **Season**, **Summary**, **Anime Notes**, **Opening**, **Ending** and **Air Date**.

In each episode there is a Long Summary and a Short Summary. However, since the Short Summary is a subset of the Long Summary, it was decided to only keep the Long Summary, unless no Long Summary was found, where we would then collect the Short Summary instead.

As for the subtitles, the operations to collect the episode script consist in downloading the subtitle file and then parsing the subtitles present in each one.

3.4 Processing operations

After collecting the data, it is required to do some data verification, cleaning and preparation, so that it is in better condition for the following steps.

To verify the data, it was required to print all of the results and check for anomalies or outliers. After finding out an anomaly, proper action must be made. In the case of the anomaly not presenting some field of data that is not crucial, then that attribute can be considered as null during the scraping. However, if one of the main attributes weren't present after the scraping process, then some filtering would have to be done to that episode in particular.

During this part, we found many inconsistencies that had to be repaired:

- **Romanization**: One inconsistency we immediately noticed was between the usage of "Alabasta" and "Arabasta", especially in the episode summaries. The decision comes down to translation choices and romanization practices. The intended spelling is "Alabasta" but because the Japanese pronunciation is closer to "Arabasta" some translations ended up using that as the spelling. For consistency purposes, and also to follow the official translations, we chose to replace every instance of "Arabasta" with "Alabasta". The same reasoning was used for other words with the same problem.
- **Same arcs being counted as different**: This problem occurs due to the anime's need of "filler episodes", meaning extra episodes that don't follow the manga and that aren't necessarily canon. These episodes exist to let the manga progress and add distance between the manga and the anime, because while both manga chapters and anime episodes are released weekly, one anime episode can adapt up to 4 chapters of the manga. Therefore, there are some small filler arcs released in the middle of a bigger, canon arc. In order to handle this, the wiki classifies the bigger arcs as continuations after the filler arc. For consistency and better visualization of the data, we decided to merge these "continuation" arcs into their main, respective arc.
- **Incorrect writing**: We noticed some data was being classified as different due to extra spacing or different capitalization. We simply had to standardize this in order to make our data consistent.
- **Fixing dates**: Due to **Pandas** not recognizing our air date column as a date, we had to use **Pandas'** function *to_datetime* and convert it to a recognized date format.
- **Unaired episode**: The wiki already has some data regarding the next episode to be aired, such as its title. However, it is missing the key information that is the summary, therefore, we removed it from our dataset.
- **Subtitle Variations**: The episodes' subtitles were too inconsistent, since they also included subtitles of the openings and endings, as well as recaps and previews, which is information that is not relevant to the current episode. To overcome these obstacles, a time based range was applied to each episode, so that it skips the opening, recap, ending and preview. This time based range was defined differently for different subsets of episodes, since the durations of openings, endings, previews and recaps vary from a group of episodes to another.

3.5 Conceptual Data Model

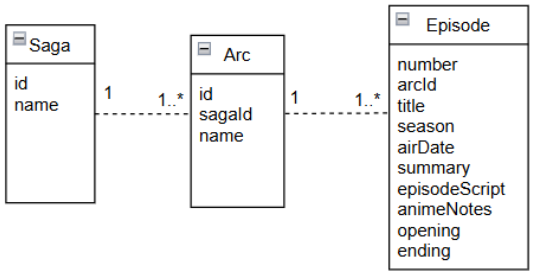

Figure 3: Conceptual Data Model

Figure 3 shows the Conceptual Data Model for the database after the collection and processing operation of the pipeline. In this format, it is easy to view the interactions between the Sagas, Arcs and Episodes of the show. Each Saga can contain 1 or more arcs and each arc can contain one or more episodes. The more in depth information is located in the Episode table such as the episode summary and the episode script.

Afterwards, after the data collection and processing, the results are structured into a **CSV** file. The **CSV** represents the Conceptual Data Model in a tabular format, where every entry represents each episode of the show and relevant information pertaining to it, including the saga and arc where it appears.

4 Characterization

4.1 Collection Characterization

As established earlier, our data is composed of 1120 episodes, this dataset is composed of many different data types that makes this a complete dataset such as:

- **Numerical**: Episode and Season number
- **Categorical**: Arc, Saga, Opening and Ending
- **Date**: Air Date
- **Text**: Title, Summary, Anime Notes, Episode Script

Our dataset covers episodes that aired from 1999 until the latest episode at the time of writing this. On average, there were 43 episodes, 2.7 arcs, 1.9 seasons and 1.4 sagas per year. The year with the most episodes was 2014, where 50 new episodes came out. Despite this, the year with the most seasons, arcs and sagas was 2001

with 4 seasons, 8 arcs and 2 sagas. The biggest gap between episodes was between episode 929 and 930 with a 70 day gap. These episodes were released on April 19, 2020 and June 28, 2020 respectively. This gap was due to the ongoing COVID-19 pandemic [16]

4.2 Descriptive and Exploratory Statistics

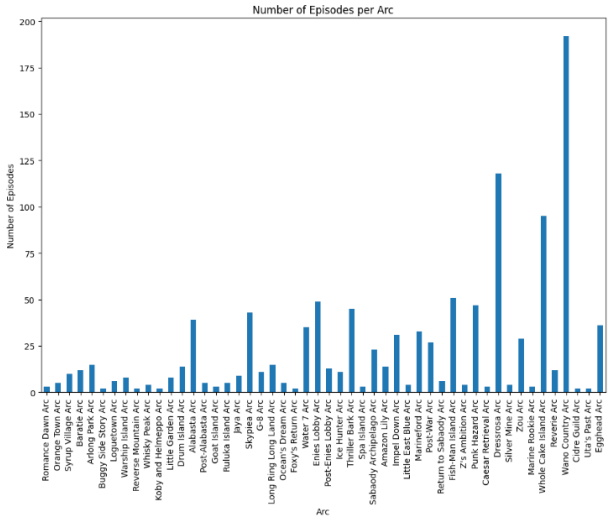

Figure 4: Episodes per Arc

Figure 4 shows that the arc with the most episodes was the Wano Country arc with 192 episodes, which had 2 filler arcs in between. However, the average number of episodes per arc stands at 22.4.

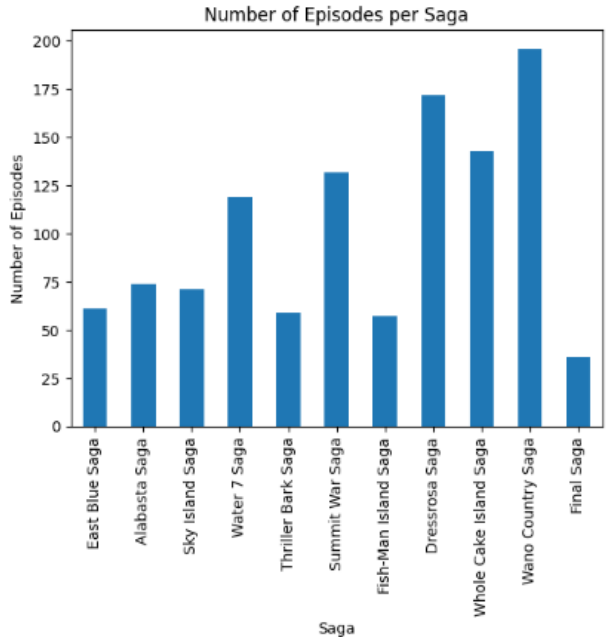

Figure 5: Episodes per Saga

Figure 5 shows that there were, on average, 102 episodes per saga, with the most episodes on a saga being on the Wano Country Saga with 196 episodes, which encapsulates the biggest arc. This saga only has 2 other arcs, them being filler ones with 2 episodes each. This tells us about the great relevance and importance of this saga and arc to the One Piece universe. Despite this, it is not the Saga with the least amount of arcs. That title belongs to the current Saga, the Final Saga, with currently only one arc. On average there are 4.5 arcs per saga and the one with the most amount of arcs was the East Blue Saga with 8.

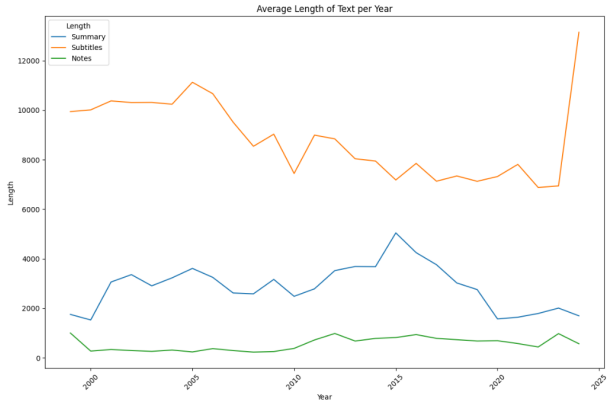

Figure 6: Length of Text per Year

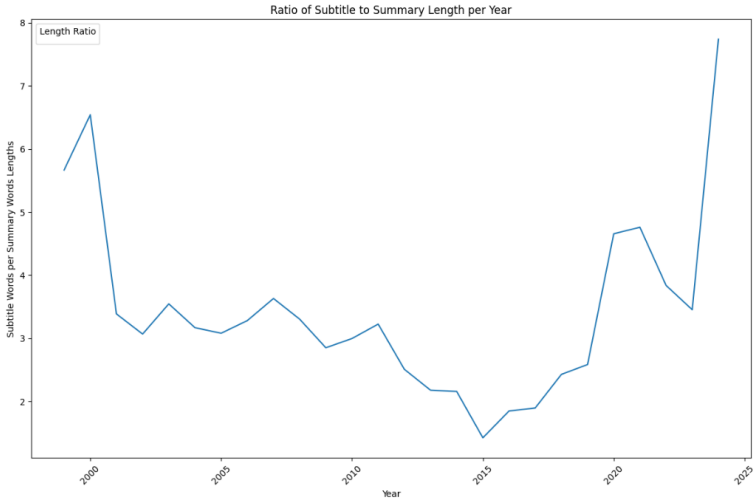

Figure 7: Ratio of Subtitle to Summary Length per Year

Figure 6 demonstrates the evolution of the lengths of text that can be found in the larger text fields of the data source. The subtitles naturally have a longer length, since it represents the entirety of the content of the episode and the anime notes have the lowest, since it only features special detail that the viewer might have missed. One thing to notice is the apparent decrease in length of subtitles over the years, until the year 2023 where it radically increased and in the year 2020 the summary length decreased substantially.

Figure 7 points to an explanation of why the sudden increase of words in subtitles has maintained the small amount of words in the summary. Over the most recent years and arcs, One Piece has been developing episodes that are more slow paced and not as relevant to the main storyline, therefore for the same or higher amount of subtitles, less than expected summary words are needed to describe the whole episode. This could become relevant since more irrelevant content bloats the search system in the amount of good results that it can retrieve.

4.3 Text Analysis

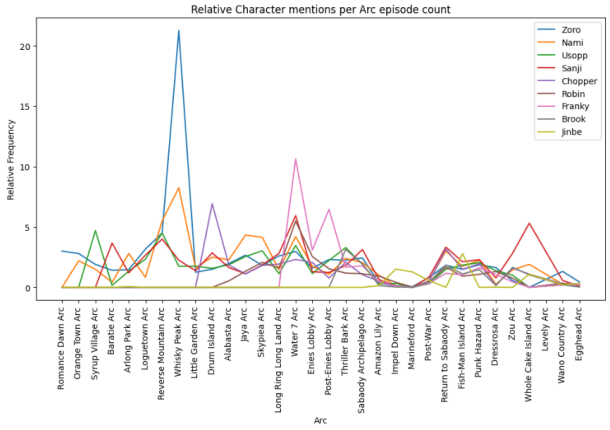

Figure 8: Ratio of Character Mentions of Episodes of each Arc

Figure 8 shows the frequency on which each of the main characters of the show are mentioned in each episode's summary. However, the final result is the ratio of the mentions of all episodes in each Arc. With this, we can correlate Named Entities to a certain location in time in the show or episode. The characters that are the main driving point behind the narrative of an Arc tend to be mentioned more in it.

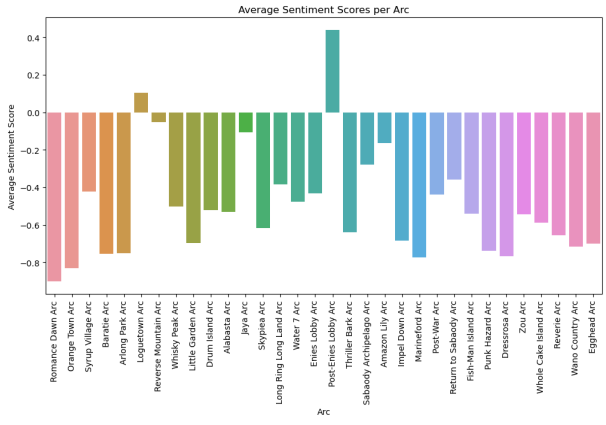

Figure 9: Sentiment Analysis of each Arc

Figure 9 shows the overall feeling of the Arc, as it takes the average of the sentiment of each episode of each Arc. Interesting to note is that, for the sole exception of two arcs (Post Enies Lobby and Loguetown), every arc is on the negative side of the sentiment spectrum. This makes sense since, generally, the main plot point of an Arc touches on a very sensitive topic, such as slavery or revolution. On the other hand, these two arcs with positive sentiment are relatively light-hearted.

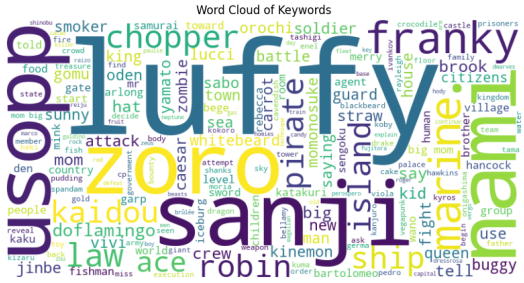

Figure 10: Summary Keywords Wordcloud

Figure 10 shows the wordcloud of the keywords inside all the episode summaries. With this, we can clearly see the main characters of the story, such as Luffy, Sanji and Zoro, as we saw earlier in the ratio of mentions. Besides this, we can also see many enemies of the straw-hats, while also seeing many words that represent the story itself.

5 Prospective Search Tasks

5.1 Description

The search tasks that the system should be effective towards should be queries that contain relevant context to the information that was collected in the pipeline and that return specific episodes.

Therefore, the user may try to perform the following queries:

- On which episodes do we learn about the straw hat pirates' childhood during the East Blue saga?
- On which episodes do the straw hat pirates have their bounties increased?
- On which episodes are any of the ancient weapons explained?
- On which episodes does Luffy duel with a straw hat pirate?

The system won't perform well for queries that require access to information that were not retrieved during the pipeline process, since this information is not present in the database, there will be almost no context associated to it. Also, the system won't support queries that require a natural and subjective result. Some example of queries that won't be able to be answered properly:

- Which animators worked on this episode?
- What was this show's writer previous projects?
- Which episodes received a good public opinion?

5.2 Information Needs

The user will perform all types of queries and each one will have an associated information need that the user requires in order to be satisfied.

To test and evaluate our system, some information needs should be presented. Some of the following examples might serve as information needs that a user might have for each of the prospective search tasks:

- **Straw Hat pirates childhoods**: The Straw Hat pirates are the main characters of the show. However, their childhood backstories are only revealed in a small number of episodes. Many users wish to know more about the characters they find most interesting and their childhoods are usually the stage where the characters traits and motivations first appear.
- **Straw Hat pirates bounty increase**: Bounties are an iconic segment of One Piece. Many people find it interesting to keep track of the value of the bounties of characters, even more if it's beloved characters like the Straw Hat Pirates. However, after knowing when it increased, it is of no use to know episodes on which they are mentioned but not increased.
- **Ancient Weapons explained**: The Ancient Weapons in One Piece are one of the most speculated topics of the show. Many people wish to discover more about them, therefore they wish to know on which episodes they are explained.
- **Luffy Straw Hat pirate duel**: People that watch One Piece tend to like the main characters and even have favorites. These characters always work and are happy together, but there are rare scenarios where a duel between them occurs. Viewers like to not only see the good sides of the main characters, but also their bad sides and disagreements since it contrasts so much with their normal selves.

6 Collection & Indexing

6.1 Document Definition

Now with our dataset complete, a document will consist of a single episode. Therefore, a single document will have stored the title, summary, anime notes, episode script, arc, saga, opening, ending, episode number, season and airdate.

6.2 Indexed Fields and Processing

As mentioned previously, all the fields scraped from the website were also added as fields in our system. For all these fields, we set as true both *indexed* and *stored*, so it is searchable and returnable.

For each of the scraped fields we also need to assign them types according to the ones we defined (*text*, *small_text*, *number* and *date*):

- **text**: This is the main field type, that has analyzers so it can tokenize and filter the text. Therefore, the most important fields with the most information use this type. The fields that use this type are the *Title*, *Saga*, *Arc*, *Summary*, *Anime Notes* and *Episode Script* fields.
- **small_text**: This field is for basic exact matching, usually placed for fields that are short and not so relevant. The fields that use this type are the *Opening* and *Ending* fields.
- **number**: This field is for numeric type content. The fields that use this type are the *Episode* and the *Season* fields.
- **date**: This field is for date type content. The field that uses this type is the *Air Date* field.

6.3 Indexing Process

To index our data and handle the varied data types in the dataset, multiple field types were defined, including text for long text fields with information that we will need to process, small_text for smaller categorical fields, number for numeric fields and date for date fields. The text field type is equipped with specific tokenization, filtering, and transformation processes to optimize both relevance and retrieval performance.

In order to process the text field, our schema defines an analyzer that, first, uses *solr.StandardTokenizerFactory*[10] in order to tokenize the text, then we apply *solr.ASCIIFoldingFilterFactory*[1] to handle non-ASCII characters, *solr.LowerCaseFilterFactory*[4] to ensure case-insensitive searches, then we built a list of synonyms and applied a *solr.SynonymGraphFilterFactory*[12]. Our synonyms are based on the One Piece universe so it can provide better results for the user. For example, the character Sanji is often refered to as Black Leg, therefore, if a user queries the system using Black Leg in its prompt, the system will correctly connect Black Leg to Sanji. Additionally, there are characters that are part of a group called "Warlords", so we added a directional synonymn so that the system relates Warlord to, i.e. Doflamingo and Buggy, but if searched for Buggy it won't have Doflamingo as a synonymn. Due to a synonym factory requirement we also added *solr.FlattenGraphFilterFactory*[3]. After that, we added a *solr.StopFilterFactory*[11] using the predefined english stop words to filter them out, and finally, we applied a *solr.PorterStemFilterFactory*[7] for stemming since our data is in english and the Porter's version is faster than the Snowball Stemmer[9].

For small and not so relevant fields, we used a *solr.StrField*, which is used for fields where the exact value is important and no text analysis is needed. We also added *sortMissingLast* as true, as there are episodes without an ending, ensuring these documents are sorted last when sorting in ascending order and *docValues* as true, enabling efficient sorting, since it may be important to sort by these fields.

For integer number attributes, we added a int point field and also set *docValues* as true, allowing efficient numeric range and sorting operations.

Finally, we added a data field type (*solr.DatePointField*) for date type fields with also *docValues* enabled for similar reasons as mentioned previously.

6.4 Schema Details

In order to evaluate the performance of the search system, we opted with the creation of 2 different schema. One that will use basic index processing and parameter settings that is called a *basic_schema* and another that will represent the current version of the search system that is called the *release_schema*. The details of the *release_schema* have already been discussed, from the index processing to the fields assignments. As such, we will explain the details of only the *basic_schema* now.

The *basic_schema* follows a very simple approach. The fields are the same and the types are the same, however the only thing that changes is the processing of the *text* type. The *text* type in the *basic_schema* uses the *solr.StandardTokenizerFactory*[10] to tokenize the text and only uses the *solr.ASCIIFoldingFilterFactory*[1] and *solr.LowerCaseFilterFactory*[4] filters. In this way, the *basic_schema*

text analyzer is less efficient and less dynamic than the *release_schema* text analyzer, which in theory severely affects the performance evaluation of the search system.

7 Retrieval

7.1 Retrieval Process

For the retrieval process, now that the data is indexed and stored, some search parameters can be set in order to improve the search while using *Solr*'s[18] query parser *edismax*[2]. The parameters remain constant throughout all of the different queries that the user can make, so some ideas had to be explored in order to tune the system into retrieving better results.

7.1.1 ***Field Boosts:*** In order to know which fields we should boost to maximize the best results within our system, we first had to analyze our fields and then test with our queries.

To do this, we used *edismax*'s query field (*qf*) and phrase field (*pf*) and boosted the fields within it. In query field we are boosting the relevance of the amount of usage of the words in said field, while in the phrase field we are boosting the relevance in which the full query appears inside a field.

We first came to a conclusion that the fields that should receive greater boosting were small and very relevant ones such as *Saga* and *Arc*, giving it a boost of 50 in *qf*. This is due to the fact that searching for a specific *arc* or *saga* should give results of mainly the episodes in that *arc* or *saga*. We didn't add a boost inside *pf* as an usual query will search for more information rather than just "East Blue Saga" i.e., as that search wouldn't have the need for a complex search system.

Afterwards, we noticed that our field *Summary* contained very pertinent data, whilst also keeping it not too long, therefore we added a medium-high boost of 30 in *qf* and 10 in *pf*.

Then, fields with a bigger text, but with scattered information that is not consistently good were placed with a medium-low boost value, such as *episode_script* (10 in *qf* and 5 in *pf*) and *anime_notes* (15 in *qf* and no boost in *pf*).

7.1.2 ***Search Flexibility:*** In order for the system to better recognize the user's queries and not be limited to what was exactly written, we added parameters that allow for some flexibility within the user's query such as *minimum match* (*mm*), *query slop* (*qs*) and *phrase slop* (*ps*).

The *mm* parameter in *Edismax* tells the system how many query terms must match within a document for it to be considered relevant. By adding this for specific values, when the number of terms in a query increases, the proportion of terms that must match decreases. This way, we allow shorter queries to ask for stricter matching and giving longer ones, which are usually more complex, the ability to retrieve partial matches.

The *ps* parameter was set to a value of 5, allowing some flexibility in the proximity of terms within a field for phrase matching. This is especially useful when handling queries where the word order might not align exactly with the indexed data, which happens a lot within the *Summary*. For example, the query "Luffy bounty raised" should still match if in the summary it finds "Luffy and his crew were told his bounty has been raised." This approach captures natural variations in phrasing within fields like *Summary* and *episode_script*.

The *qs* parameter, set to 2, ensures similar flexibility for multi-term matches in the query itself, allowing documents to score higher when query terms appear close together in the text. This is particularly effective when dealing with complex queries that might naturally contain slight variations in word placement or structure.

7.1.3 ***Additional Parameters:*** Beyond flexibility and boosts, we added other parameters that improve our system's performance, such as the query operator (*q.op*), which ensures all text elements of the query must be present in a episode for it to be considered relevant. This is particularly relevant as terms like "Luffy" and "Straw Hat pirates" are very common and relevant query items and are very common among all documents. This operator is also balanced by the use of flexibility as mentioned previously, ensuring relevant results appear while also disregarding irrelevant ones.

Another critical parameter we added was the *tie* parameter. This parameter, which determines how much influence a match in a less-relevant field (based on its boost value as described previously) contributes to the overall score of a document, which was set to 0.1. By setting a low value, the system ensures that matches in higher-boosted fields, like *Saga*, *Arc*, or *Summary*, dominate the ranking, while still allowing minor contributions from less-critical fields to improve nuanced relevance.

Finally, we add *stopwords*, ensuring these don't contribute to the documents score and *lowercaseOperators*, so that when the query contains an "or" or "and", they will be treated as logical operators, making the system more user-friendly, since many might not know about the existence of these terms.

After exploring all of these ideas, 2 parameter sets were developed. One default that we can use as a comparison reference and another that is used as the final version of the system, using everything we've mentioned previously.

7.1.4 ***default parameter set:***

(1) *defType*: dismax;
(2) *rows*: 30;
(3) *qf*: Title Arc Saga Summary episode_script anime_notes

7.1.5 ***final parameter set:***

(1) *defType*: edismax;
(2) *rows*: 30;
(3) *qf*: Titleˆ10 Arcˆ50 Sagaˆ50 Summaryˆ30 episode_scriptˆ10 anime_notesˆ15;
(4) *q.op*: AND;
(5) *mm*: 7<80% 10<60% 15<50%;
(6) *pf*: Summaryˆ10 episode_scriptˆ5 anime_notes;
(7) *ps*: 5;
(8) *qs*: 2;
(9) *tie*: 0.1;
(10) *stopwords*: true;
(11) *lowercaseOperators*: true

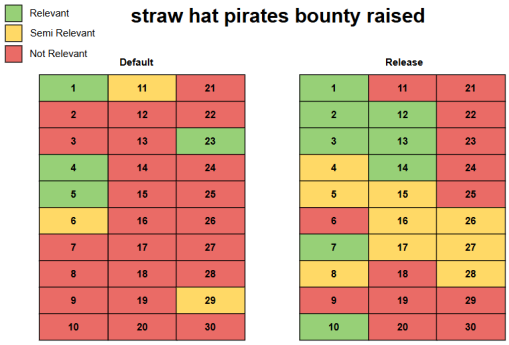

Figure 11: Demo comparing the default version with the release version. Each rectangle represents a document that was retrieved by the query present in the title

7.2 Demo

8 Evaluation

8.1 Information Needs Reviewed

In the previous milestone, the information needs of the system or user were discussed. These information needs are directly tied to the evaluation of the search results. The information needs of the user breaks down into 3 main requirements:

- Guarantee a high precision to recall rate;
- Guarantee a high precision and recall average;
- Guarantee the relevancy of the top searched results.

As for the information the user may want when using the system, it's mostly about topics that aren't mentioned much or that are very dispersively mentioned along the anime. Not only that, but also topics related to the main characters or the main storyline, that are subjects that users are always eager to learn more about.

Some key needs that the user might have are: Obtaining the episodes of the Straw Hat pirates childhoods, the episodes where the Straw Hat pirates bounties have increased, episodes where the ancient weapons are explained and the episodes where Luffy duels a straw hat pirate.

8.2 Comparison of Setups

For each information need, a query was made but in 2 different systems, one using the basic schema with the default parameters and the other using the release schema with the release parameters. The same queries were made for each of the information needs, keeping in mind that extra details might be added to the query only to reduce the set of possible outcomes into a subset (i.e. only results from a certain saga):

- **IN1**: straw hat pirates childhood east blue saga;
- **IN2**: straw hat pirates bounty raised;
- **IN3**: ancient weapons explained;
- **IN4**: Luffy duels straw hat pirates

A clear difference between the systems can be seen when comparing the results of each of these IN. The release version measures the accuracy and precision results almost 4 times better on average than the default version. Also, in terms of relevancy, the release version presents more relevant documents at the beginning in comparison with the default version that seems to present inconsistent results that don't match the user's expectations.

Now, let us take a look at the average of the precision metrics for both systems for each IN and the corresponding precision-recall curves:

	MP3k	MP6k	MP9k	MAP	MAUC
Default	0.17	0.2	0.17	0.1496	0.1561
Release	0.75	0.46	0.39	0.5685	0.5845

Table 1: System Metrics

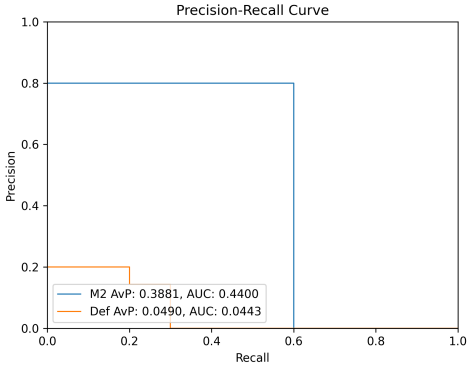

Figure 12: Precision-Recall Curve of the Release System for the IN1

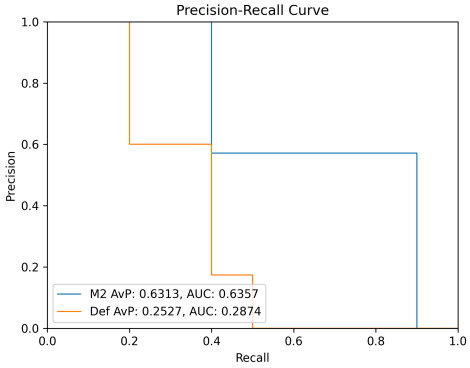

Figure 13: Precision-Recall Curve of the Release System for the IN2

8.3 Manual Evaluation Process

To perform the manual evaluation, a pipeline process was used, the queries, qrels, parameters for each IN are placed in a json file and a pipeline python process will parse this file, query solr and generate results which it can use to compare with the qrels in each

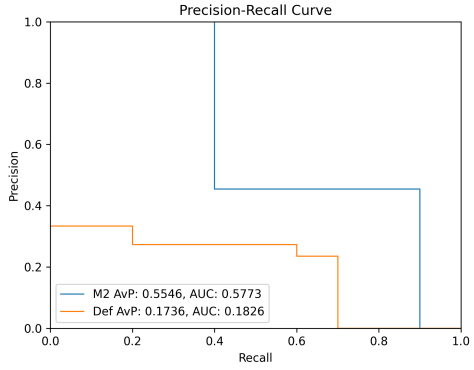

Figure 14: Precision-Recall Curve of the Release System for the IN3

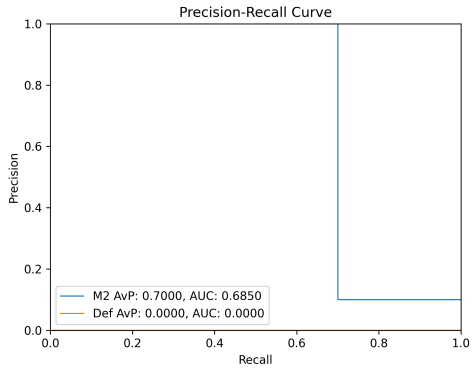

Figure 15: Precision-Recall Curve of the Release System for the IN4

IN, which can then be used to generate diagrams to measure the system's performance. The qrels chosen for each IN were picked by hand, using the One Piece Wiki[6] as reference.

Regarding the query about the episodes in which the straw hat pirates' bounty got raised, we noticed that in a few episodes the straw hat pirates discover that they have a new bounty or talk about it, despite the fact that the viewer already knew about it. For example, the summary of episode 152 has sentences such as "Usopp confirms that it was a new bounty poster and informed Luffy and Zoro of their amounts", among others that talk about their bounties. Therefore, the system struggles to disregard these episodes as irrelevant. This query also misses one episode due to the fact that it is not directly mentioned that Luffy's bounty has been raised, but it is implied instead.

When it comes to the query regarding the episodes on which we learn about the straw hat pirates' childhood during the East Blue saga, we noticed that the system excludes episodes that don't include the words "backstory", "childhood" or "past", even though the episodes talk about a character's past, because it is implied, making it miss 3 correct episodes. It also wrongly matches with episodes where anything related to their childhood happens, despite not giving the viewer new information about their childhood, such as in episode 48: "Zoro helps Tashigi, but accidentally crushes her glasses when he discovers she bears a striking resemblance to his childhood friend".

Regarding the query about the episodes on which any of the ancient weapons are explained, we notice some episodes that wrongly match due to the fact that they talk about the ancient weapons, but they are not explained. This is seen, for example, in episode 570: "Robin mentions that there are three ancient weapons: Pluton, Uranus, and Poseidon.". Like the other queries, it misses one episode because the ancient weapons being explained are implied. Not only that, but some ancient weapons are characters, such as Shirahoshi, and if we add it as a directed synonym so the system recognizes her as an ancient weapon, it only worsens the performance.

Finally, regarding the query about episodes on which Luffy battles a member of the straw hat pirates crew, our system wrongly matches some episodes where there is talk about a duel happening among Luffy and members of the crew, such as episode 235, when it leads to the actual duel in episode 236 and when a member of the crew is involved in a duel, despite not being against Luffy or vice versa. Despite finding all the required documents, it struggles to match when the duel is only implied and not directly mentioned.

8.4 Discussion

The results of the search system are overall very positive. It is a very good baseline search system, however, there are certainly aspects to improve. The main strengths of this search system are the filters with special attention to the synonyms, carefully thought out search parameters and the high amount of scraped data that allows the system to run with better precision. In terms of weaknesses, it suffers a lot on certain queries. Since not all the synonyms are in place yet, some queries will perform significantly worse, especially if it becomes too specific or subjective. Another weakness would be the non use of embeddings, since they can capture the meaning, positions and context of the phrase. This would lead to a better match to queries where the correct result is only implied within the document, as described during the manual evaluation process. As a result, the addition of more relevant synonyms and embeddings is the next logical step to increase the search system performance.

9 Improvements

After developing 2 systems, further techniques were explored in order to improve the performance of the searching results. There are new ideas that can be implemented that might raise the accuracy of search results, but there are also many things that can be improved or previous aspects of the search system that could be developed a bit further.

9.1 GUI Interface

One of the most lacking aspects of the previous version of the search system was the lack of a GUI interface to allow the user to search for the information in a user-friendly way. To solve this, we developed a website using React for the front-end and the Django Python Framework for the back-end.

The main page provides the user with a simple search bar which can be used to query the search system for information regarding

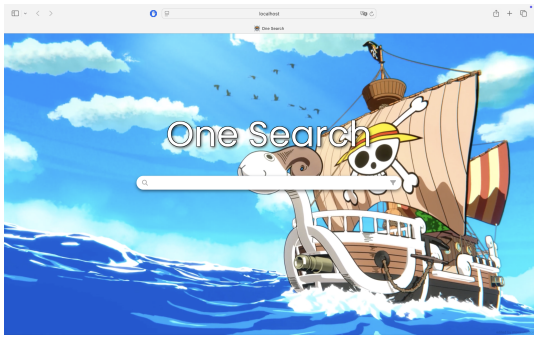

Figure 16: Website's Main Page

the One Piece anime. Additionally, seeing as the results returned by the system represent episodes and episodes are part of an Arc and Saga, the user may want to search for information regarding a specific Arc or Saga. Therefore, the website provides a simple filtering system, where the user can select which Arcs and/or Sagas he wants to search for. This filters list was not added manually to the site, instead it is retrieved from the *data.csv*, which contains the information of all the scraped episodes. If a new episode is scraped which belongs to a new Arc or Saga, it will automatically be added to the filters list.

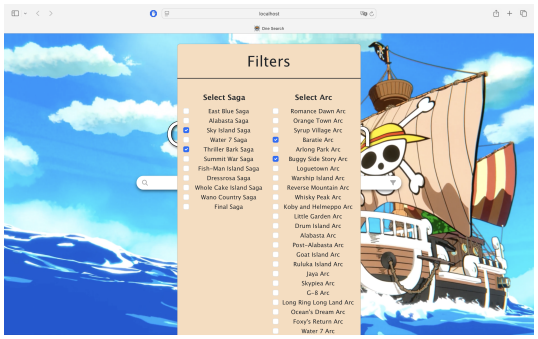

Figure 17: Search Filters

When a user searches for something, one of two things can happen: either there were episodes found for that query, after which they will appear on the main page, or no episodes were found and a warning regarding that is shown to the user.

With episodes found, the user can click any one of them and be directed to a new page, specifically for that episode, with more information.

This page provides more information about an episode, something that the small cards on the main page don't show, such as the air date of the episode, anime notes, the rest of the episode's summary, etc. Moreover, the title of the webpage changes from "One Search" to "Episode n | One Search" to indicate the specific episode's page. Not only that, but the URL also changes to have "episode/<episode_id>" appended to the website's URL. This allows the user to retrieve information about a specific episode without using the search system in the main page. Finally this page provides the user with a back-arrow to go back to the main page. To improve

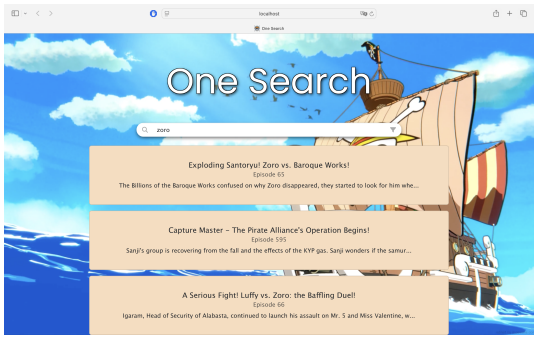

Figure 18: Episodes found for user's query

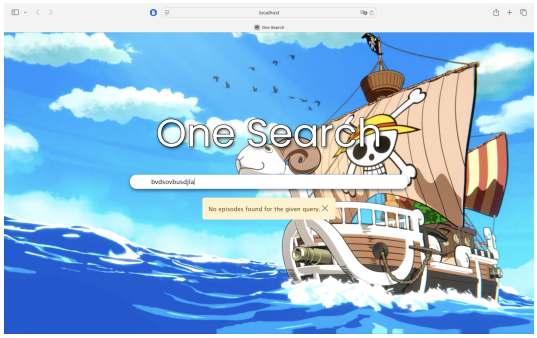

Figure 19: No episodes found for user's query

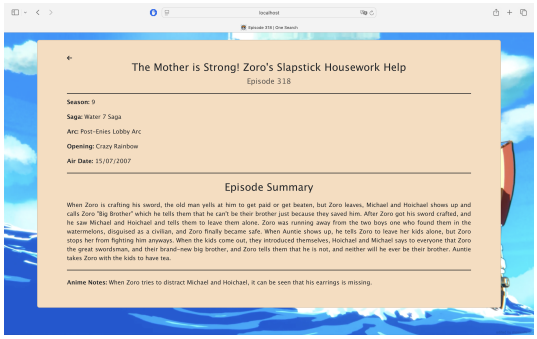

Figure 20: Episode Page

user-experience, after returning to the main page, the previous search results will be maintained, along with the correct pagination number. Finally, the user can reset the search results by just clicking the big "One Search" title on the main page.

9.2 Embeddings

The main problem of the search system currently, is that it doesn't really grasp the context of the document that it is fetching. In other words, whenever a document is matched with the query and solr parameters, it is just doing standard matching and filtering of tokens. This process leads to a very linear search system, as we visualize matching of documents as the matching of specific tokens in them. In reality, when a user wants to search for a specific

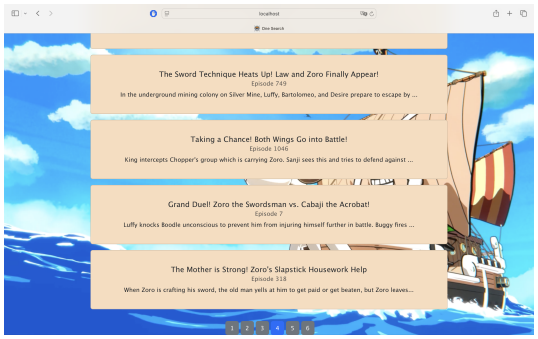

Figure 21: Main Page after clicking the back-arrow

document, he tends to want to search for a particular context in the document or a certain meaning that cannot be matched with simple one on one token analyses.

The most obvious solution to this problem is the implementation of embeddings or vector embeddings.

Note that if we wish to turn our data into vectors then we can not go past the vector dimensions. This brings another challenge into turning our data into vector embeddings.

9.3 Vector Embeddings Creation

So we now know that it is useful to turn our data into vector embeddings, and we know what the schema requires in order to use these embeddings in our search. However there are still 2 steps required to make this happen.

One of the steps is to turn our data into vector embeddings. But, what part of the data should we turn into embeddings? How much of that data should be embeddings? How do we create these vectors?

An initial approach was to turn all of the data into embeddings. After doing so, the dimensions of the vectors of each document were significantly higher than the vectorDimensions imposed in the schema. Furthermore, another problem was that a lot of the scraped information wasn't the most useful to use in terms of context. We then decided to reduce the amount of data to turn into embeddings. We focused solely on turning the summary into embeddings, since the summary of episodes contains the most relevant and short sized information of our data.

But the problem still remains, the summary contains too many tokens and if we try to embed all of these tokens, it will truncate in most cases due to the all-MiniLM-L6-v2 limitations, which is the model we're using.

To fix this problem we embed several vectors of subsets of the summary. That way we have a set of vectors which can be used for a document. Further approaches like doing the mean of all these vectors to combine them into one, or sum of them and normalization could be used. However, due to rigorous testing of different cases, using all of these vectors to identify the same document proved the most efficient. To do this we could simply have an array of vectors and each vector represents chunks of the summary. We also normalized all our vectors in order for solr to use the dot product algorithm, which is faster and retrieves the same results as cosine.

The division of the summary in chunks is made by splitting the text whenever a ".", "!", "?" or other sentence ending punctuation marks appear. That way the context of the vectors can be better represented, since normally the context of a phrase is dependent on itself and not previous or future phrases.

Since now we have an array of embeddings for each phrase in a summary, *csv* format was no longer usable due to the variable number of amount of vectors per episode, also, it would become too cluttered. Therefore, we made the conversion to *JSON* and, for each episode, there's an array called *vectors* with many *vector* fields with the embeddings.

9.4 Embeddings Indexing

To index all these vectors we created a new field called *vector*. This new field will create new documents for each vector added and set it as child document of the episode it is from.

To handle this new field we created a new field type called *denseVector*. In this field type, *solr* will turn the vector input into a *solr.DenseVectorField*.

The algorithm we used in **denseVector** is **HNSW** which essentially builds a set of nodes where each node represents a document which has an inner matrix with the nodes that are similar to him based on the similarity function. That way to find similar nodes one can simply traverse the graph, and the purpose is to cluster the documents in a graph data structure that way it performs the search faster just by traversing the already constructed graph, instead of performing the similarity function with each document in the set. The similarity function to form the matrices inside each document node is the dot product as mentioned before, since it's a faster version of cosine similarity.

The **HNSW** algorithm can have a number of max connections and a beam width. The *hnswBeamWidth* allows to select a number of node candidates that a new graph node will have. For our schema, we found out that the best results were achieved when this was declared as 200.

The *maxConnections* parameter will indicate how many of those candidates will actually become the node's nearest neighbor. In our case, the best result was achieved with 32 max connections.

Additionally, the *DenseVectorField* requires a *vectorDimension* number which will be used to define the number of dimensions in a vector that is embedded, which in our case is 384, as specified by the model we've used and previously mentioned.

Finally, we can make queries that search for similarity in any one of the vectors inside the vector field and return the parent document. That way for all the vectors created for a given summary, they will all be children of the same episode document.

9.5 Querying with Embeddings

Now that we have all the embeddings ready and indexed, we must add a way for the user to use them in the query.

In order to do this, we had 2 options to explore: either use only the embeddings and let them give the correct documents to the user, or use them to re-rank the system we built previously.

Using only the embeddings is rather simple. First, we transform the query into embedding format normalized, since we are using dot product. Then, we add the following to the q parameter: *{!parent*

which=":* -_nest_path_:*" score=avg}*. This will make sure the average score calculated with the vector document will be redirected to the parent. Finally, we add *{!knn f=vector topK=520}* followed by the query embedding, which will apply the KNN algorithm to the vector field using top K as specified. The KNN algorithm is ideal since it returns a k amount of nearest neighbors for that query.

The second option, re-ranking, isn't as simple, but is easily doable. In order to do this, we added a new field to the parameters called *rq*. This new field, specified as *{!rerank reRankQuery=$rqq reRankDocs=40 reRankWeight=95}* will use the field *rqq* results to rerank the previously given documents based on the amount of *reRankDocs* specified with the weight specified in *reRankWeight*. Now, inside *rqq* we place the same we did for the *q* in the embeddings only system.

9.5.1 Parameter Finetuning. To ensure our values for the parameters needed in this new queries, *score*, *topK*, *reRankDocs* and *reRankWeight* are the best, we developed a *Grid Search* that tried out many different values and combinations and choosing the one with the highest average between the MAP and median AUC value, which got a rounded value of 0.60 using all INs. The best values were the ones previously mentioned as used in the parameters fields.

9.6 Embeddings Evaluation

With this two new systems in place, we now have to evaluate how they perform both against each other and against the other systems we've previously developed.

To do this, we used our previously mentioned pipeline and added this two new systems.

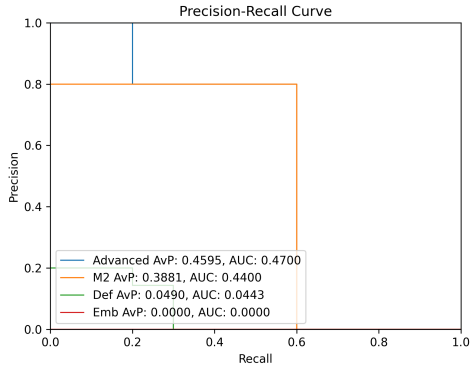

Figure 22: Final Precision-Recall Curve of the Release System for the IN1

Based on the previous analysis of the previous results and now comparing with the new results in 22, 23, 24 and 25, where Advanced represents the newly improved system with re-ranking and Emb represents the new embeddings-only system, we can conclude that, by adding embeddings to our system, it can be slightly improved, but not as much as we might've expected, even worsening at times, i.e. IN4.

We can also see that, by themselves, embeddings are quite useless, managing to get a score of 0 on almost every IN.

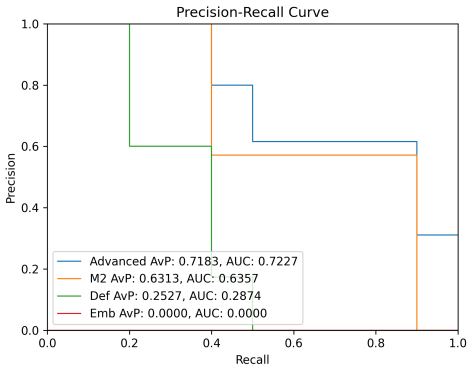

Figure 23: Final Precision-Recall Curve of the Release System for the IN2

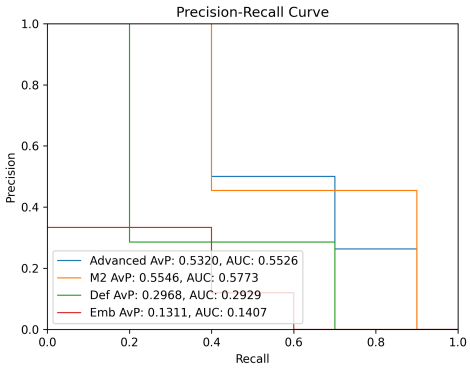

Figure 24: Final Precision-Recall Curve of the Release System for the IN3

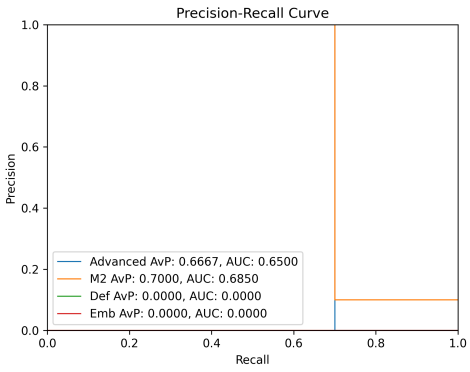

Figure 25: Final Precision-Recall Curve of the Release System for the IN4

This worse performance might be due to the fact that the model which generated the embeddings might not have that much information about the *One Piece* universe in order to correctly correlate

and calculate the embeddings distance. This idea is also reinforced by the fact that using only embeddings the scores are even worse than the basic system we developed as a baseline.

In 27, 28, 29, 30 we can see the predictions that were accurate over k documents. This adds on to the previous statements, since the results are in line with the precision-recall curves that were analyzed. However, it is always useful to have the evaluations in graph form in order to visualize the comparisons of systems in a better way.

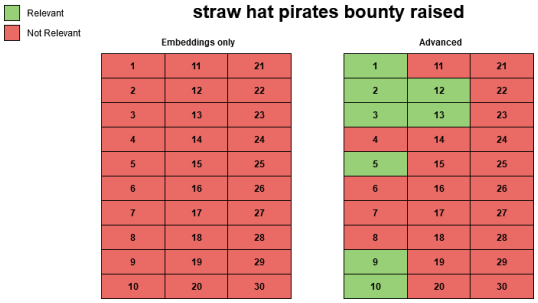

Figure 26: Demo comparing new systems

In 26 we added our new systems to compare relevancy of results. This time we also didn't use the "Semi-Relevant" metric for simplicity sake and assumed these previously "Semi-Relevant" episodes as not relevant. As we can see, this new advanced system improves on the previous achieved results in 11

	MP3k	MP6k	MP9k	MAP	MAUC
Default	0.17	0.2	0.17	0.1496	0.1561
M2 Release	0.75	0.46	0.39	0.5685	0.5845
Embeddings Only	0.08	0.08	0.06	0.0328	0.0352
Advanced	0.75	0.5	0.42	0.5941	0.5988

Table 2: System Metrics

9.7 Discussion

To conclude, the overall end results were satisfying, but due to limitations regarding the fact that some functions can't perform as good as they should since our system is based on the *One Piece* universe, such as the embeddings model, the results were not as good as we might've wanted to.

10 Annexes

References

[1] [n. d.]. *ASCII Folding Filter.* https://solr.apache.org/guide/solr/latest/indexing-guide/filters.html#ascii-folding-filter
[2] [n. d.]. *Edismax Query Parser.* https://solr.apache.org/guide/solr/9_0/query-guide/edismax-query-parser.html
[3] [n. d.]. *Flatten Graph Filter.* https://solr.apache.org/guide/solr/latest/indexing-guide/filters.html#flatten-graph-filter
[4] [n. d.]. *Lower Case Filter.* https://solr.apache.org/guide/solr/latest/indexing-guide/filters.html#lower-case-filter
[5] [n. d.]. *One Piece English Subtitles.* https://subdl.com/sc/one-piece-complete-series-wan-piisu-one-piece/english
[6] [n. d.]. *One Piece Wiki.* https://onepiece.fandom.com/wiki/One_Piece_Wiki

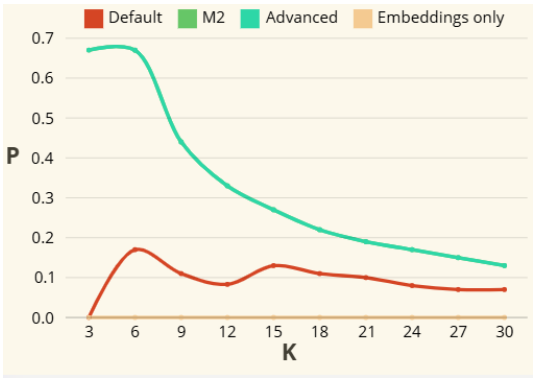

Figure 27: Final PK Curve of the Release System for the IN1

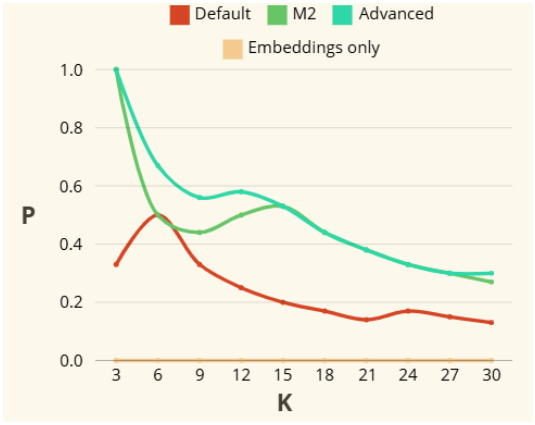

Figure 28: Final PK Curve of the Release System for the IN2

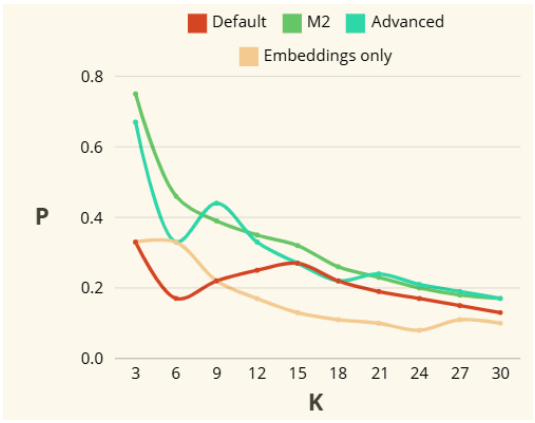

Figure 29: Final PK Curve of the Release System for the IN3

[7] [n. d.]. *Porter Stem Filter.* https://solr.apache.org/guide/solr/latest/indexing-guide/filters.html#porter-stem-filter
[8] [n. d.]. *React.* https://en.wikipedia.org/wiki/React_(software)
[9] [n. d.]. *Snowball Porter Stemmer Filter.* https://solr.apache.org/guide/solr/latest/indexing-guide/filters.html#snowball-porter-stemmer-filter
[10] [n. d.]. *Standard Tokenizer.* https://solr.apache.org/guide/solr/latest/indexing-guide/tokenizers.html#standard-tokenizer

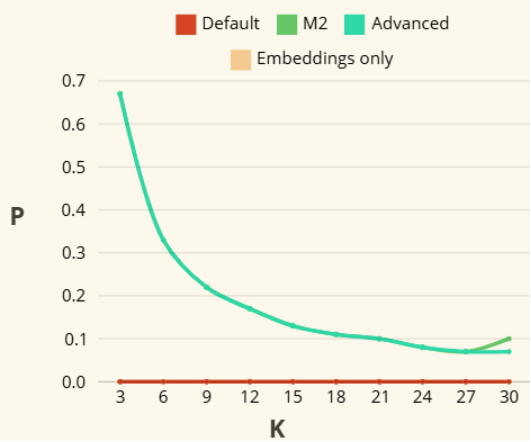

Figure 30: Final PK Curve of the Release System for the IN4

[11] [n. d.]. *Stop Filter.* https://solr.apache.org/guide/solr/latest/indexing-guide/filters.html#stop-filter
[12] [n. d.]. *Synonym Graph Filter.* https://solr.apache.org/guide/solr/latest/indexing-guide/filters.html#synonym-graph-filter
[13] Brano. [n. d.]. *One Piece English Subtitles.* https://www.opensubtitles.org/pb/ssearch/sublanguageid-pob,por/idmovie-22054
[14] Brano. [n. d.]. *Opensubtitles License.* https://www.opensubtitles.org/pb/disclaimer
[15] Samuel Dauzon, Aidas Bendoraitis, and Arun Ravindran. 2016. *Django: web development with Python.* Packt Publishing Ltd.
[16] Christine King. 2020. *One Piece anime on indefinite break due to COVID-19 pandemic.* https://www.nag.co.za/2020/04/20/one-piece-anime-on-indefinite-break-due-to-covid-19-pandemic/
[17] Leonard Richardson. 2007. *Beautiful soup documentation.* https://readthedocs.org/projects/beautiful-soup-4/downloads/pdf/latest/
[18] Yonik Seeley. [n. d.]. *Solr's Web Page.* https://solr.apache.org/