Tunix, a Music Search System

António Pereira
Faculty of Engineering, University of Porto
Porto, Portugal
up202205501@up.pt

Nuno Rios
Faculty of Engineering, University of Porto
Porto, Portugal
up202206272@up.pt

Rodrigo Resende
Faculty of Engineering, University of Porto
Porto, Portugal
up202108750@up.pt

Rodrigo Ribeiro
Faculty of Engineering, University of Porto
Porto, Portugal
up202206396@up.pt

Abstract

The rapid growth of digital content demands robust information retrieval systems capable of indexing and searching large volumes of unstructured text. Music is a particularly relevant domain: although audio–fingerprinting services excel at matching audio snippets, there is a gap for systems that support search using *textual* evidence such as lyrics and artist descriptions. This manuscript documents the first milestone of *Tunix*, a music search system centered on text. We (i) identify, acquire and integrate a public lyrics dataset with external artist metadata, (ii) design a reproducible preparation pipeline, and (iii) provide a systematic characterization to ground future search functionality.

CCS Concepts

• **Information systems → Query representation**; *Data extraction and integration*; • **Computing methodologies** → *Natural language processing*.

Keywords

information retrieval, text mining, music search, lyrics, data preparation, exploratory data analysis

ACM Reference Format:
António Pereira, Nuno Rios, Rodrigo Resende, and Rodrigo Ribeiro. 2025. Tunix, a Music Search System. In *Proceedings of (G73).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Tunix is a music search system that allows users to explore songs and artists using rich text attributes such as lyrics, emotions, and similarity to other tracks and artists.

Text–based search over musical content enables use cases that audio–only systems cannot address, e.g., finding songs about a theme, tracing an artist's career through biographical narratives, or filtering lyrics by writing style. In this project we focus strictly on text and refrain from semantic or affective search beyond keyword and statistical evidence, as per the project plan.

Although services like Shazam are known to match audio to songs, there is a notable gap in music search systems based on text attributes–few systems are capable of searching for tracks and artists based on descriptive information, such as lyrics, *mood* and similarity. This project aims to explore and develop a system to address this gap.

2 Data Sources

We adopt a publicly available dataset of songs containing titles, artist names and lyrics, obtained from Kaggle[1]. Although informally referred to as the "Spotify Million Song Dataset" on its landing page, the distribution we use contains tens of thousands of tracks across several hundred artists. To complement the lyrics with richer artist context, we query the Last.fm API[2] to retrieve short artist biographies and album titles when available. All web resources are properly cited using BibTEX entries of type `@misc`.

The resulting raw working dataset comprises four attributes from the original distribution (song title, artist name, lyrics, and a link field) and two attributes extracted via API calls (artist biography and album title). The external link field is not required for our search scenarios and is therefore discarded during preparation.

Table 1: Basic statistics of the original distribution (before enrichment).

Feature	Artists	Songs
Kaggle lyrics distribution	643	57,650

Notes on provenance and credibility. Kaggle hosts the dataset and provides basic provenance information; the original collection is programmatically derived from the Spotify API according to its description. Last.fm biographies are a mix of editorial and community–contributed content; to mitigate sparse or missing entries we filter out cases that lack substantive text (see Section **??**).

3 Data Preparation

The data preparation process begins by downloading the original dataset from Kaggle, provided as a CSV file. The first task is to clean

[1]See [1] for dataset landing page.
[2]API documentation in [2].

the dataset, which involves identifying and removing any empty or null values in the attributes, as well as eliminating duplicate rows. Fortunately, we did not encounter any such issues, so we did not remove any rows. This is the key reason we selected this dataset.

Although the collection only contains four features, one of them–the *link* attribute–was redundant for our purpose. It is an external URL pointing to the song on another website. We do not need this reference, so the attribute was dropped.

After cleaning the Kaggle dataset, we proceeded to scrape the biography of each artist and album name of each track from the Last.fm API. To achieve this, we first obtained an API key by signing up on the Last.fm website and registering an application. The key is necessary to authenticate requests and access the API data.

Within the course of this project, we relied on the following Last.fm API endpoints, which return data ion the JSON format:

(1) `track.search(track_name, artist_name)`
(2) `track.getInfo(track_mbid)`
(3) `artist.getInfo(artist_mbid)`

We prepared `(track_name, artist_name)` pairs for each row to be used as input for the initial set of API requests. In doing so, we standardized the capitalization and spacing of the names to ensure consistency and avoid potential query mismatches. We then used endpoint (1) with each pair to retrieve the Music Brainz identifier–*mbid*–for each song, to be used in subsequent requests.

With the unique identifier for each track, we then utilized the `track.getInfo` endpoint to obtain the album titles and the *mbid* of the artists, discarding all other metadata from each response. We encountered 39 missing album titles, which were attributed to tracks not associated with an album, likely standalone singles. To address this, we filled in the missing album titles with *No Album*.

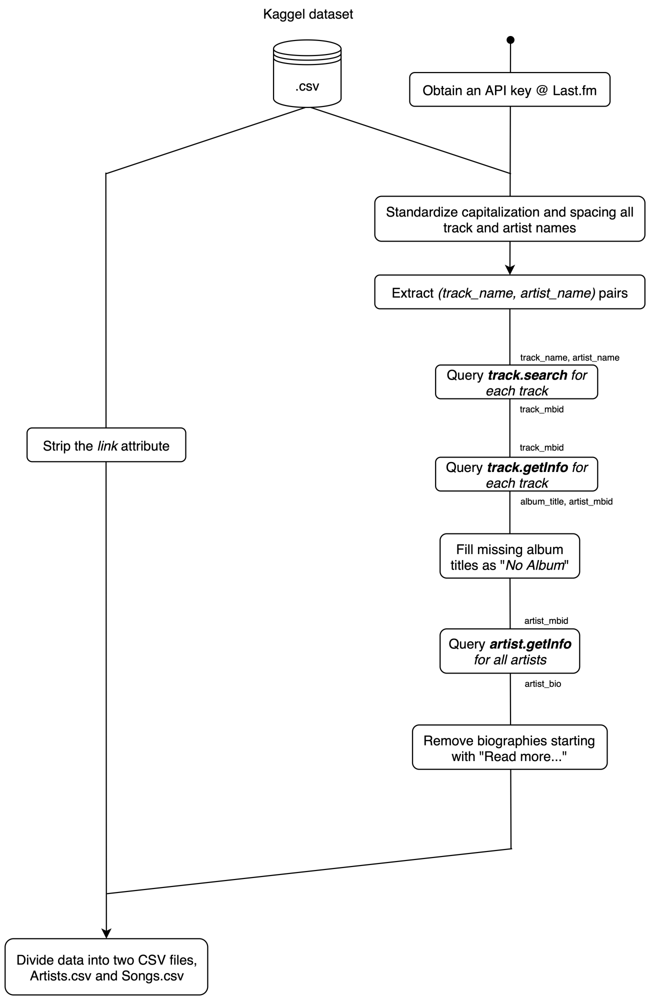
Next, we queried `artist.getInfo` using the artist *mbids* obtained earlier, to retrieve a short biography for each artist in our dataset. We encountered 35 instances where the biography was essentially a prompt to "Read more on Last.fm", indicating that the website did not provide a detailed biography for these artists. As a result, and since it was quite a small count, we decided to remove these rows from the dataset, leaving us with a final count of 608 artists, as shown in Table 2.

Table 2: Basic statistics of the initial vs final dataset.

Dataset	Features	Artists	Songs
Spotify Million Song Dataset	4	643	57650
Tunix Dataset	5	608	55185

The final dataset was stored in two separate CSV files—one for the songs and another for the artists. We opted for two files rather than one to avoid repeating the biography of each artist in multiple rows, as each artist has an average of 89 songs. Storing all data in a single file would have been inefficient, both in terms of data redundancy and unnecessary file size growth. Additionally, we chose the CSV format over a more specialized SQL-like format, as the data model is relatively simple. We also added a unique id for each song and artist, starting from 1.

Figure 1: Data Preparation Diagram.


Kaggel dataset
.csv
Obtain an API key @ Last.fm
Standardize capitalization and spacing all track and artist names
Extract (track_name, artist_name) pairs
track_name, artist_name
Query track.search for each track
track_mbid
track_mbid
Query track.getInfo for each track
album_title, artist_mbid
Fill missing album titles as "No Album"
artist_mbid
Query artist.getInfo for all artists
artist_bio
Remove biographies starting with "Read more..."
Strip the link attribute
Divide data into two CSV files, Artists.csv and Songs.csv

4 Data Characterization

After cleaning the data, we analyzed it to understand its structure and main features. We used Pandas, NumPy, and Matplotlib to explore and visualize the information in an insightful manner.

4.1 Conceptual Data Model

Figure 2 shows the conceptual data model depicted as a UML diagram. It is a simple model, which follows the structure of the two files of the final dataset, with one class representing an artist and another representing a song. Each song belongs to an artist, and each artist may have sung or produced multiple songs.

A song may sometimes be sung by multiple artists, but we should note that our model does not support this as its original dataset linked each song to a single artist. This is acceptable, however, as the artist listed represents the primary or primary credited artist, typically the one who holds the legal rights to the track.

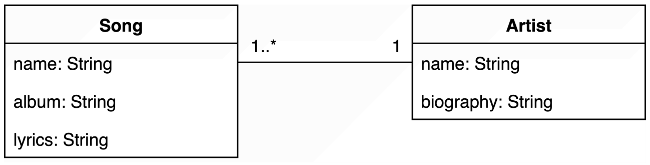


Figure 2: UML Diagram

5 Exploratory Data Analysis

5.1 Word Clouds

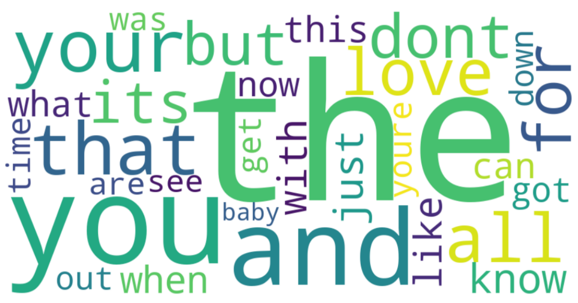


Figure 3: Most frequent terms in lyrics after stopword removal. Head terms dominate scoring; synonym handling or phrase queries may be required to diversify results.

Figure 3 displays the most common words found across all song lyrics in the dataset. The size of each word represents its relative frequency of occurrence. Common English words such as "the," "you," "and," and "your" appear most prominently, reflecting their high frequency in lyrical language. The presence of emotionally charged words like "love" and "baby" also highlights recurring themes typical in popular music.

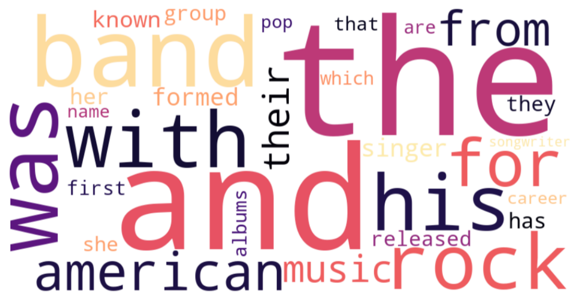


Figure 4: Frequent terms in biographies. Biographical text often covers formation, releases and nationality; useful for filters and query expansion but less discriminative than lyrics for topical search.

The word cloud shown in Figure 4 represents the most common words found in the artist biography texts. The larger words, such as "band," "rock," "american," and "music," highlight dominant themes related to genre, nationality, and professional context. Frequent terms like "released," "formed," and "career" indicate that the biographies often describe the formation of bands, musical styles, and the evolution of artists' careers.

5.2 Lyrics Length Distribution

Figure 5 illustrates the distribution of the number of words in song lyrics across the dataset.

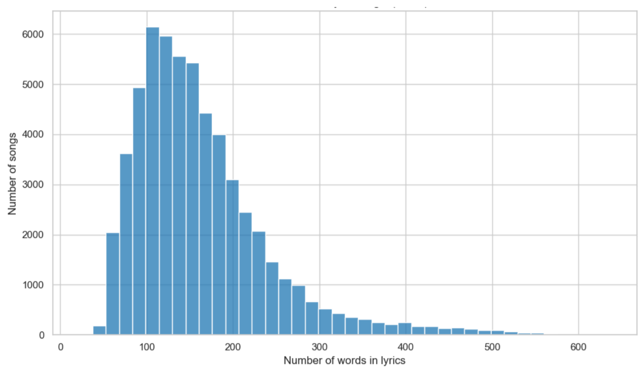


Figure 5: Distribution of lyrics length (tokens). Tune length normalization during ranking and consider minimum length thresholds for certain queries.

The analysis of lyric length helps characterize the dataset in terms of document size and complexity. The distribution shows how many words each song contains, indicating variability in lyrical structure. Longer lyrics may correspond to more elaborate narratives, while shorter ones suggest simpler or more repetitive compositions.

5.3 Lyrics Word Frequencies Distribution

The distribution of word frequencies in the lyrics follows Zipf's Law, as shown in the Zipf plot (rank vs frequency). This indicates that a small number of words are very frequent, while most words appear rarely, a typical pattern of natural language.

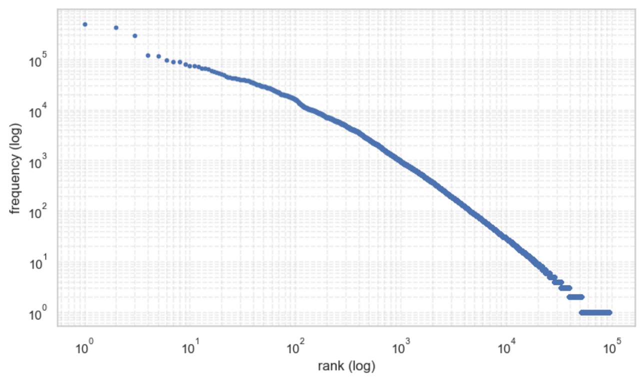


Figure 6: Zipf plot for lyrics. Reinforce IDF weighting and consider phrase queries for improved precision.

6 Propective Search Tasks

These objectives are intended to ensure that the search system offers meaningful and context-aware results. By addressing both functional and interpretative needs, the system will enable users to explore songs and artists in a more intuitive and analytical way, beyond simple keyword matching.

We formulate realistic Information Needs (INs) as **questions**, relying primarily on unstructured text fields — *song lyrics* and *artist biographies*. Categorical fields are used only as filters.

IN1 — How can we find songs that express specific emotions, such as love or anger, through their lyrics?

Motivation. Emotion-based search supports mood-centric discovery (e.g., love songs).
Approach. Query `song_lyrics` with emotion keywords and synonyms (WordNet expansion), rank with BM25; optionally filter by artist/album.
Example. `q = love OR "in love" OR heartbreak OR romantic`.
Evaluation. Relevant if the dominant sentiment in the lyrics matches the target emotion.

IN2 — How can we retrieve songs similar in lyrical content or theme to a given query song?

Motivation. Discover songs sharing themes with a known track.
Approach. Use the lyrics of a query song as a pseudo-query (BM25/TF–IDF over `song_lyrics`); optional filters by artist/period.
Example. Terms from *"Heartbreak Hotel"*: `heartbreak, lonely, crying, room, hotel`.
Evaluation. Relevant if lyrics show thematic/semantic similarity to the seed song.

IN3 — How can we identify songs that contain explicit or violent language?

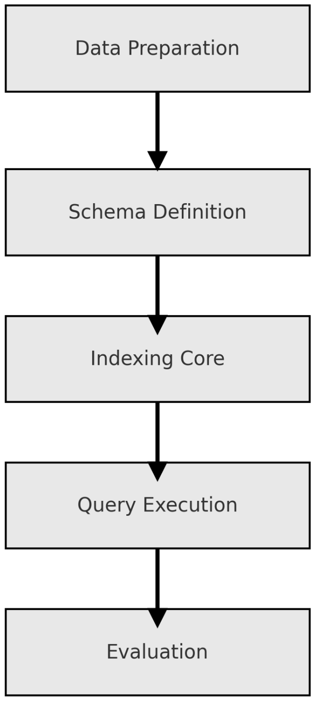
Motivation. Enable parental controls or content-sensitive filtering.
Approach. Define a lexicon (e.g., `kill, gun, blood, damn, hate, fight`); query `song_lyrics`; compute a per-document intensity score and apply thresholds.
Example. `q = kill OR fight OR blood OR hate OR damn`.
Evaluation. Relevant if explicit/violent vocabulary is present in the lyrics (precision@k at early ranks).

IN4 — How can we relate an artist's background to the lyrical themes in their songs?

Motivation. Biographies provide context that can align with lyrical topics.
Approach. Query both `artist_bio` (e.g., "American rock band") and `song_lyrics` (e.g., "freedom", "rebellion") with field boosts; use country/genre filters if needed.
Example. Find American rock artists with lyrics about freedom.
Evaluation. Relevant if background cues in `artist_bio` are consistent with themes in `song_lyrics`.

7 System Overview

Figure 7: System architecture: ingestion, schema definition, indexing, querying, and evaluation workflow.



The **Tunix** system was implemented as a modular information retrieval pipeline using Apache Solr 9, deployed inside a Docker container. The architecture integrates five key stages: data ingestion, schema definition, indexing, querying, and evaluation.

The overall workflow is as follows:

(1) **Data Preparation:** Preprocessed CSVs generated in Milestone 1 serve as input data.
(2) **Schema Definition:** The schema defines custom text analyzers and field types for English text.
(3) **Indexing:** Documents are ingested into a Solr core named `songs`.
(4) **Querying:** Queries are executed using the EDisMax parser with fuzzy, wildcard, and proximity configurations.
(5) **Evaluation:** Results are converted to TREC format and evaluated using `trec_eval`.

The process begins with preprocessed files generated in Milestone 1, containing fields such as *song_id*, *song_name*, *artist_name*, *album_name*, *song_lyrics*, and *artist_bio*. These documents were ingested into a Solr core.

At query time, users can search through any indexed text field (e.g., lyrics or biography). The query is parsed using Solr's default text analyzer for English.

8 Document Analysis

8.1 Schema

The Solr schema was designed to accommodate both structured metadata and unstructured text fields (lyrics and biographies). Each document corresponds to a single song, and the fields are defined as shown below:

- **id (string)** — unique identifier, indexed and stored.
- **song_name (text_general)** — full-text searchable field for titles.
- **song_lyrics (text_lyrics)** — main field for retrieval; uses tokenization, lowercasing, stopword removal, and stemming.
- **album_name (string)** — exact match, used for filtering and faceting.
- **artist_name (string)** — exact match, useful for grouping and filtering.
- **artist_bio (text_bio)** — full-text searchable biographical information.

The analyzers used in the *text_lyrics* and *text_bio* field types combine the `StandardTokenizerFactory`, `LowerCaseFilterFactory` and `StopFilterFactory`. This configuration ensures that English lyrics and biographies are tokenized properly, made case-insensitive, stripped of stopwords, and stemmed for improved recall.

8.1.1 StopWords. We curated a custom list of stopwords beyond the default English stopwords. This list includes common words and pronouns that are not useful for distinguishing song topics (e.g., “the”, “and”, “oh”, “yeah”, etc.). By updating the schema to use this stopword list in the text analysis chain for lyrics and other text fields, we ensure these terms are removed during indexing. This reduces index size and prevents common words from diluting the relevance of query matching. For example, words like “again” or “alone” which might appear frequently are filtered out so that the search focuses on meaningful terms.

The field types `text_lyrics` and `text_bio` use analyzers composed of:

- `StandardTokenizerFactory`
- `LowerCaseFilterFactory`
- `StopFilterFactory`
- `SnowballPorterFilterFactory`

This ensures robust tokenization, case-insensitive matching, and effective stemming for English text.

8.2 Synonym Expansion

A synonym generation module was implemented using the NLTK WordNet interface (`synonyms.py`). It processes all unique words from lyrics and biographies, removes punctuation and stopwords, and queries WordNet to identify synonym sets.

We integrated two kinds of synonyms into the indexing process:

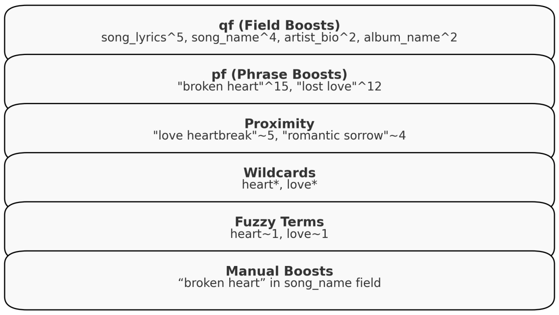
- `(1) Manual synonyms:` a domain-specific set of equivalent terms and phrases that we compiled (in `synonyms_hand.txt`). These cover informal expressions or thematic terms related to our queries. For instance, for the concept of heartbreak, we included slang or colloquial terms like “broke up”, “dumped”, “lonely”, etc., which might not appear in a generic thesaurus but are relevant in song lyrics.
- `(2) WordNet synonyms:` synonyms obtained from WordNet for a broader vocabulary. We generated a large list of synonyms (`synonyms_output_wordnet.txt`) by extracting synsets for many terms. For example, WordNet provides synonyms for “heartbreak” such as “heartache” and “grief”51†. We merged these with the manual list to create a comprehensive synonym file (`synonyms.txt`) used by Solr.

The generated synonym map (`synonyms_output_wordnet.txt`) can be integrated into Solr's schema as a synonym filter for semantic query expansion. This enhances recall by enabling equivalence between words such as “heartbreak”, “heartache”, and “sorrow”.

9 Retrieval

This section outlines how Solr queries were constructed to address the defined Information Needs (INs), leveraging field boosts, phrase boosts, proximity, wildcards, and fuzziness. Two configurations were evaluated: Setup 1 (baseline) and Setup 2 (enhanced), corresponding to `1.json` and `2.json` respectively.

Figure 8: Example of Solr query parameters and boosting strategies in IN1 .


qf (Field Boosts)
song_lyrics^5, song_name^4, artist_bio^2, album_name^2
pf (Phrase Boosts)
"broken heart"^15, "lost love"^12
Proximity
"love heartbreak"~5, "romantic sorrow"~4
Wildcards
heart*, love*
Fuzzy Terms
heart~1, love~1
Manual Boosts
“broken heart” in song_name field

9.1 Field and Phrase Boosting

We used Solr's Extended DisMax parser with the `qf` parameter to define field-specific boosts. In Setup 1, lyrics were given the highest weight (`song_lyrics`^{5}), while artist biographies, song titles, and album names were each weighted at ^{2}. Setup 2 increased the weight of `song_name` to 4, based on the observation that many relevant songs reflect their theme directly in the title.

Additionally, the `pf` parameter was used to boost exact or near-exact phrase matches. Common expressions such as `"broken heart"` and `"lost love"` were boosted, especially in fields like `song_name` and `song_lyrics`, where phrase matches are likely to indicate thematic relevance.

9.2 Proximity, Wildcards, and Fuzziness

Setup 2 incorporated proximity queries (e.g., `"love heartbreak" 5`) to reward documents where conceptually related terms appear within a short distance. This is particularly useful in lyrics, where poetic phrasing often separates related ideas.

Wildcard queries (e.g., `love*`, `heart*`) were included to match multiple word forms or suffixes. Fuzzy queries (e.g., `heart~1`) added tolerance for minor lexical variations, capturing additional relevant results that may not exactly match the base terms.

9.3 Manual Term Boosting

While Solr allows independent boosting via the `bq` parameter, we opted to manually tune weights in `qf` and `pf` to emphasize especially descriptive terms and fields. For example, "broken heart" received a higher boost in titles than other terms, improving the rank of documents whose titles directly matched the query theme.

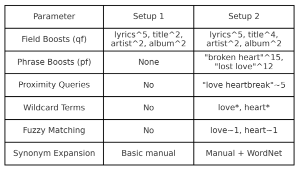
9.4 Query Construction Workflow

Each IN was translated into a Solr query using curated vocabularies sourced from synonym files (`synonyms.py`, `emotional_heartBreak.txt`, etc.). Setup 1 queries used core thematic terms only, while Setup 2 added contextual and synonymous terms, along with wildcard, fuzzy, and proximity operators. This structured approach helped balance recall and precision across both retrieval configurations.

10 System Evaluation

We evaluated two distinct retrieval setups corresponding to the queries discussed above. Setup 1 (baseline) uses limited synonym expansion and basic boosting, whereas Setup 2 (enhanced) uses extensive synonym expansion plus wildcards/fuzzy and more aggressive boosting. We performed a comparative evaluation on a set of Information Needs related to heartbreak-themed songs.

Figure 9: Comparison between Setup 1 and Setup 2 across query parameters and boosting strategies.



Parameter	Setup 1	Setup 2
Field Boosts (qf)	lyrics^5, title^2, artist^2, album^2	lyrics^5, title^4, artist^2, album^2
Phrase Boosts (pf)	None	"broken heart"^15, "lost love"^12
Proximity Queries	No	"love heartbreak"~5
Wildcard Terms	No	love*, heart*
Fuzzy Matching	No	love~1, heart~1
Synonym Expansion	Basic manual	Manual + WordNet

10.1 Methodology

We ran both setups' queries on the Solr index and collected the top results. Setup 1 was configured to retrieve up to 100 results, while Setup 2 retrieved 50 (it turned out 50 results were sufficient to find all relevant docs for the expanded query). The outputs were saved in TREC run file format (as `trec_run.txt`), listing for each query the ranked list of document IDs and scores. We then used `trec_eval` (NIST's standard IR evaluation tool) to compute metrics, comparing the run against the qrels.

10.2 Evaluation Pipeline

The evaluation follows the TREC methodology, automated through Python scripts:

(1) Manual relevance judgments (`qrels/`) converted to TREC format with `qrels2trec.py`.
(2) Solr output converted to TREC runs via `solr2trec.py`.
(3) Both evaluated with `trec_eval`, and visualized using `plot_pr.py`.

10.3 Performance Metrics

Figure 10: Comparison of evaluation metrics for Setup 1 (baseline) and Setup 2 (enhanced).

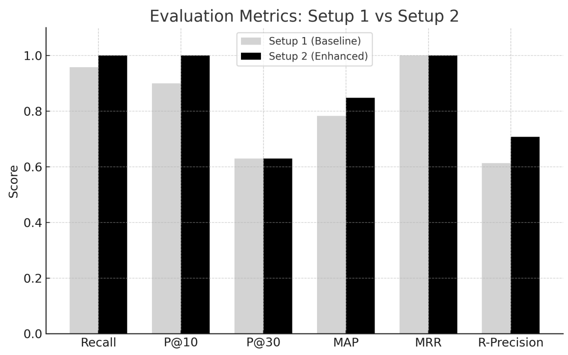


Table 3: Main evaluation results from TREC evaluation.

Metric	Query 1	Query 2	Mean
MAP	0.7828	0.8477	0.8152
R-Prec	0.7083	0.6129	0.6606
BPref	0.9583	1.0000	0.9792
P@10	0.9000	1.0000	0.9500

The Mean Average Precision (MAP) of 0.8152 indicates strong overall retrieval effectiveness, with high precision at the top ranks (P@10 = 0.95). The system consistently returned relevant documents among the top results for both queries, confirming that the text processing and tokenization settings in the schema were appropriate for the English lyrics domain.

10.4 Manual Evaluation Process

To support quantitative evaluation, we manually assessed the relevance of Solr's retrieved results for each Information Need (IN). For each IN, we reviewed the top 50–100 retrieved songs and annotated them as relevant or non-relevant based on lyrical content and alignment with the thematic intent of the query.

Judgments were guided by keyword presence, emotional tone, and conceptual alignment with the query (e.g., songs reflecting heartbreak or romantic betrayal). This process was performed independently by two annotators and later reconciled into a single `qrels.trec` file.

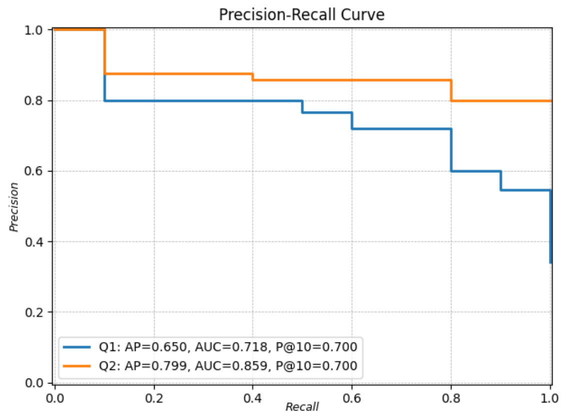
In total, 55 documents were marked as relevant across all queries. These relevance judgments served as ground truth for trec_eval, enabling us to compute MAP, P@K, MRR, and R-Precision. Manual annotation was critical for capturing nuanced cases (e.g., metaphorical lyrics) that automatic keyword matching might overlook.

10.5 Results

The evaluation results reveal that the system performs particularly well for queries involving frequent or semantically rich terms such as "love", with high recall and precision at early ranks. The BM25 scoring model effectively balances term frequency and document length, ensuring fair ranking across lyrics of different lengths.

Figure 11 presents the precision–recall curve obtained from the evaluation. The curve demonstrates stable precision across most recall levels, with a gradual decline near full recall, typical of systems optimized for top-ranked precision.

Figure 11: Precision–Recall curve for two test queries.



10.6 Observations

- The BM25 model ($k_1 = 1.2$, $b = 0.75$) provides consistent performance.
- Frequent emotional terms (e.g., "love", "heartbreak") yield the best retrieval scores.
- Ambiguous words slightly reduce precision, motivating future semantic expansion.
- Integration of synonym mapping into Solr's pipeline is expected to improve recall in Milestone 3.

11 Conclusions/Discussion

The data collection and preparation stages proved to be crucial for ensuring the quality and consistency of the final datasets. By combining data from the Spotify Million Song Dataset with information retrieved from the Last.fm API, we created a comprehensive dataset that includes both lyrical and biographical dimensions. The structured dataset and well-defined objectives will guide the upcoming implementation and evaluation phases, where the focus will shift from data preparation to search functionality.

- Full Solr-based indexing and retrieval pipeline implemented.
- Automated evaluation workflow with TREC compatibility.
- Synonym expansion using WordNet for recall enhancement.
- Quantitative validation with MAP > 0.8.

References

[1] Kaggle. 2025. Spotify Million Song Dataset (lyrics). https://www.kaggle.com/datasets/notshrirang/spotify-million-song-dataset. Online; accessed 11 November 2025.

[2] Last.fm. 2025. Last.fm API Documentation. https://www.last.fm/api. Online; accessed 11 November 2025.