Exploring the Relationship Between Musical Features and Public Sentiment on Social Media.

Jacob 'Jack' Singer
jsinger9@binghamton.edu
Binghamton University, SUNY
Binghamton, New York, USA

Leora Dallas
ldallas1@binghamton.edu
Binghamton University, SUNY
Binghamton, New York, USA

Brandon Ciaravino
bciarav1@binghamton.edu
Binghamton University, SUNY
Binghamton, New York, USA

Jason Vitale
jvitale5@binghamton.edu
Binghamton University, SUNY
Binghamton, New York, USA

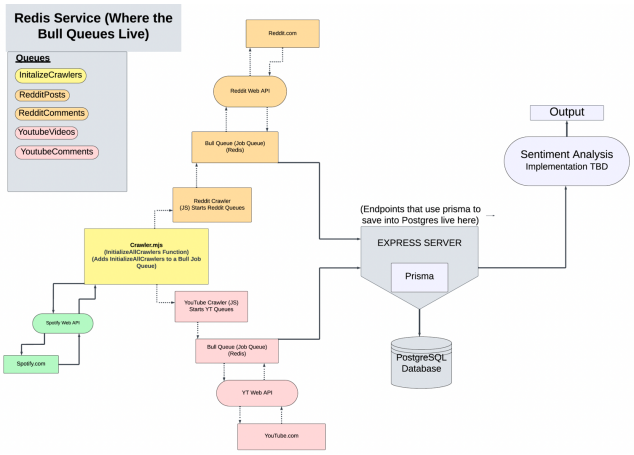

Figure 1: Pipeline Design.

1 MAIN ASPECTS OF PROJECT ARCHITECTURE

- **Database (Using Prisma ORM)**
- **Queing System and Endpoints (Using BUll)**
- **Web Server (Express)**
- **How it is all contained and deployed (Docker)**

2 DATABASE

We are using a Postgres SQL database, and we are interacting with this database using Prisma. After each crawl, an endpoint in our Express Web Server is hit and it adds to the database.

Database Table: -RedditPost, RedditComment , Subreddit, Youtube-Video , Youtube Comment, SpotifyArtist, SpotifyTrack, Spotify-TrackAudioFeatures,SpotifyAlbum, SpotifyPlaylist, SpotifyTracksIn-Playlist, Query, QueriesOnRedditPosts, QueriesOnYoutubeVideo

3 QUEUING SYSTEM AND ENDPOINTS

3.1 Crawler.mjs

Crawler.mjs directs all the other crawlers Crawler.mjs grabs the tracks from the spotify playlist that is chosen. It creates a query type object for each track which is sent to the saveQuery endpoint where it is saved to the database. For each track that we have we initialize a youtube crawl and reddit crawl for that query.

3.2 Reddit.mjs

When an initializeRedditCrawler job is created, it acquires an access token and adds a job to the Reddit queue with the query from crawler.mjs. The job is processed by the searchReddit function using this access token and query. The searchReddit function checks if the subreddits related to the query are indexed through an endpoint exposed by the Express server. Unindexed subreddits are then indexed and saved. Reddit posts are retrieved and saved via an Express server endpoint. The retrieveComments function is used to fetch comments from these posts, which likely includes comment content, author, and other metadata.

3.3 Youtube.mjs

The initializeYoutubeCrawler function triggers the YouTube crawling process. It creates a job in the youtubeVideoQueue with the query obtained from crawler.mjs. This job is processed by the searchVideos function, which uses the YouTube API to find video IDs related to the query. Retrieved video information is then saved to the PostgreSQL database, including details like video title, ID, and possibly other metadata. For each video retrieved, a new job is created to fetch comments for that video. These comments are obtained through the YouTube API and include details like the text of the comment, commenter information, and like counts. The comments are then saved to the database via an Express server endpoint.

4 ISSUES FACED

(1) Prior revisions of the crawlers were developed with Python, making it difficult to interact with the express server that was set up which was written in JavaScript.
   (a) This is due to the differences between dictionaries in Python and objects in JavaScript and how they work and are structured. By just using JavaScript, we can have uniform functionality in our objects.
(2) We then translated or crawler scripts from Python to TypeScript where we were experiencing many type related errors so we consequently translated our scripts to JavaScript.
(3) Installing Redis, which is a service that allows us to use a job queuing system called Bull.

(4) Bull took days to set up between rate limiting and setting up the correct number of jobs.
(5) Rate limiting from YouTube made it difficult to make changes, and then retest without running out of the units we get per day to use the Youtube API.
(6) Eventually we tested on just 2 queries by using the first 2 out of the 50 songs to minimize the number of calls we were making in order to test, however we realized that we were not correctly saving the comments on a Reddit post, only the post itself.
(7) We set up Prisma, the express server, and our crawler to handle comments and it seemed to work locally, upon attempting to run on the VM we faced numerous issues with Bull where we were unable to queue up the correct amount of jobs, if any at all.
(8) We wanted to view our database using the Prisma ORM from outside of the virtual machine, as well as our Redis Commander which allowed us to see what jobs were set up, it took a few days to figure out how to tunnel the ports on the VM so that we can view these things without having to be on a Binghamton computer or logged into the VM.
(9) We initially figured we would Schedule a CRON job through the VM's system and upon testing we could not figure out why the job would not execute, we switched over to try and schedule a CRON job from within our expres server again without any luck, eventually we realized that CRON works in UTC time and eventually got the Crawler.mjs script to trigger by modifying the VM's crontab -e script.
(10) After sorting out small issues that we faced with getting rate limited, and making sure Prisma was mapping the correct incoming response from Reddit and YouTube we ran the script successfully for 1 single day until we realized that although we were saving comments, they did not have a relation to a particular post or query in our database.
(11) We halted the system, and updated our crawlers and express server that contains Prisma so that we could successfully give relations between comments, posts, and queries.
(12) Since some songs stay on the top 50 charts for multiple days in a row, that meant that we would potentially be crawling Reddit and YouTube for the same query, which meant we would potentially see the same posts for this query. Being that our database did not handle duplicate queries we realized that in the event there was a query that we already looked for, its data would be overwritten and the comments/posts for the old query would be redirected to the new query resulting in the first one to have none. We had to update our Prisma schema to handle a many to many relation where multiple posts and comments could be assigned to different queries and vice versa. In doing this we also had to update our express endpoints that utilize Prisma to handle an upsert, which checks to make sure the query has not been in the database before, and if it has to update it.
(13) Although successful locally, our VM's were complaining about the new upsert addition, as well as the way we're adding postID's to multiple queries despite having updated our Prisma schema to handle this new many-to-many relation.

FIRST MAJOR UPDATE

5 FIXED

(1) Removed youtube comment queue. Now in the video process youtube comments are processed synchronously. This mitigated our issue where youtube comments were not being populated properly in the database.
(2) Added a queue for initilaizeAllCrawlers instead of doing a CRON job.
(3) Added Tox.mjs to test toxicity once data is collected.
(4) Within Reddit.mjs we are now able to get the toxicity score saved in our database.

6 REDDIT GRAPH

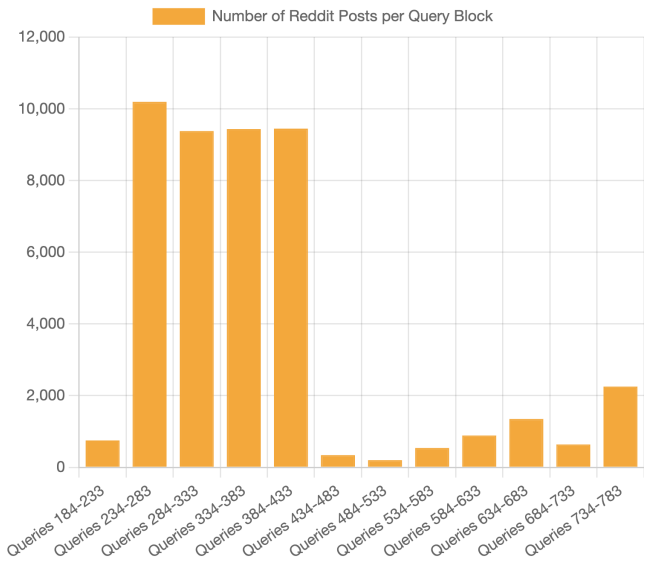

Figure 2: Reddit Daily Counts

6.1 Reddit Daily Counts

The first query reflects the first successful crawl, however we were not able to collect all the data due to how the post were managed in the database. We were able to fix this issue and get sufficient data for the next few days. We experimented adding the toxicity ratio for each batch of posts and we experienced outages and errors. However as you can see in Figure 2 the issue was resolved and we started collecting more posts.

6.2 Reddit Comments Daily Count

Figure 3 reflects the same results we saw in the Reddit posts now for the Reddit comments. At first we can see when we started to successfully collect the data in the first query. While testing the toxicity we experienced outages which is reflected in the results. However, as we resolved the issues we began collecting comments.

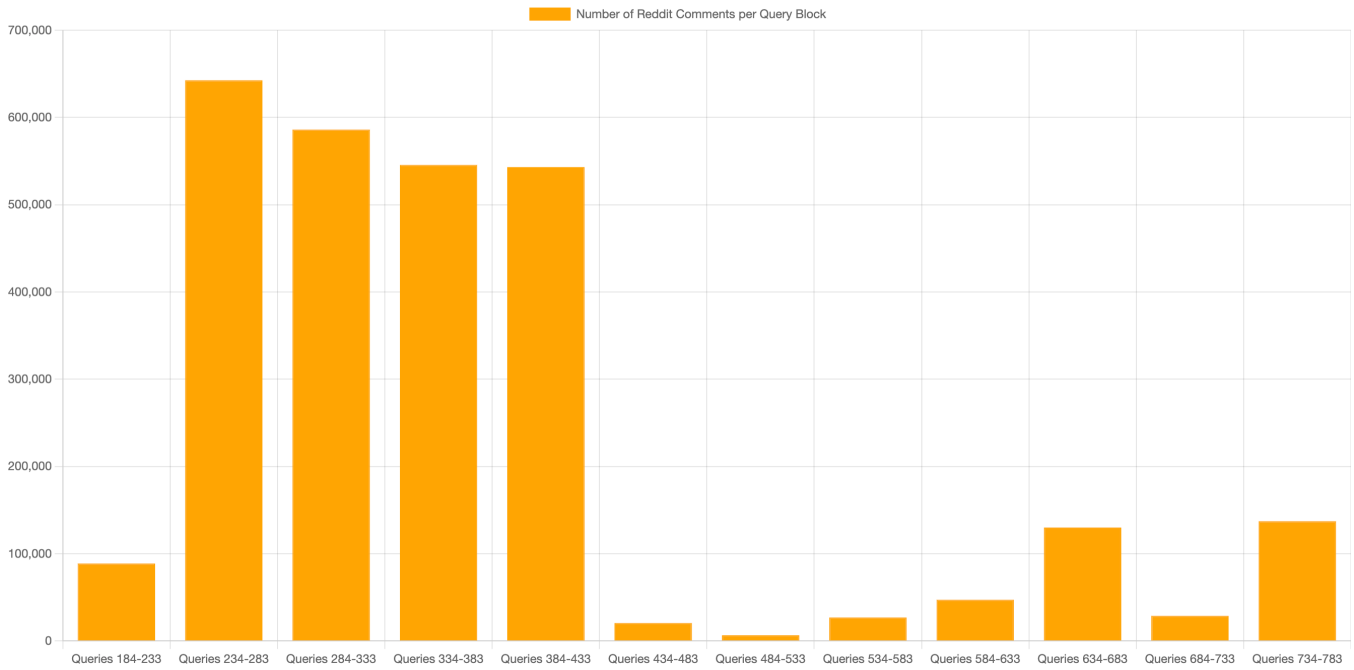

Figure 3: Reddit Daily Counts

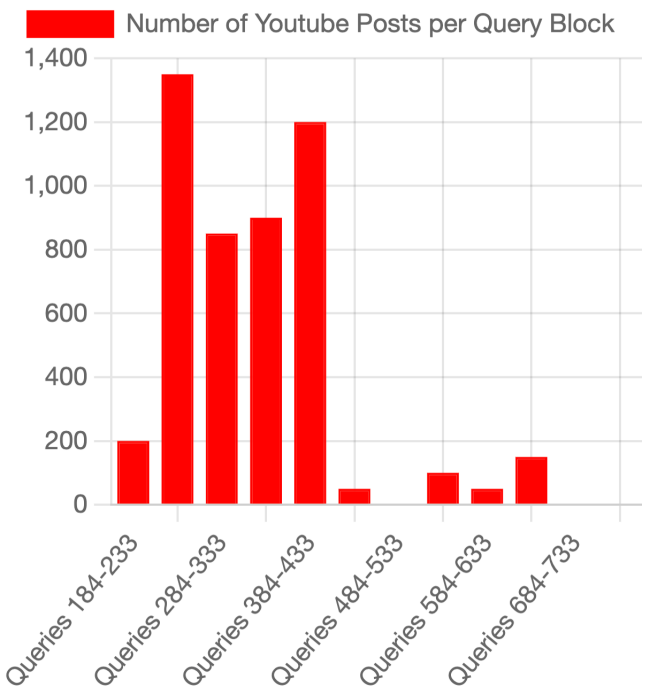

Figure 4: YouTube Daily Counts

6.3 YouTube Daily Counts

Figure 4 reflects the same issues we faced in the Reddit crawl. We were unable to collect sufficient data until we managed our posts properly in the database. Again we faced outages while integrating the toxicity,which ultimately was fixed allowing us to collect more YouTube posts.

7 NOTE

On each graph we start graphing at query number 184 because that is when the database and collection system started working, everything before that is not good data.

So far we have seen 0% toxicity from the posts we have collected thus we have not included a graph at this time.

FINAL UPDATE

8 UPDATED

We created a React Application to send dynamic queries to our endpoints where in the endpoint we return an html that gives a graph that represents the amount of data we collected for the specific graph type (Youtube Videos,Youtube Comments, Reddit posts and Reddit comments). In Figure 5 we can see the user is able to interact with buttons that will change the endpoint that is hit. The user can also change what queries will be graphed based on their Query ID. Finally the user is able to choose certain aspects of the song through Spotify's audio features. This can be done with the last 50 queries we have based on the Spotify playlist we crawl.

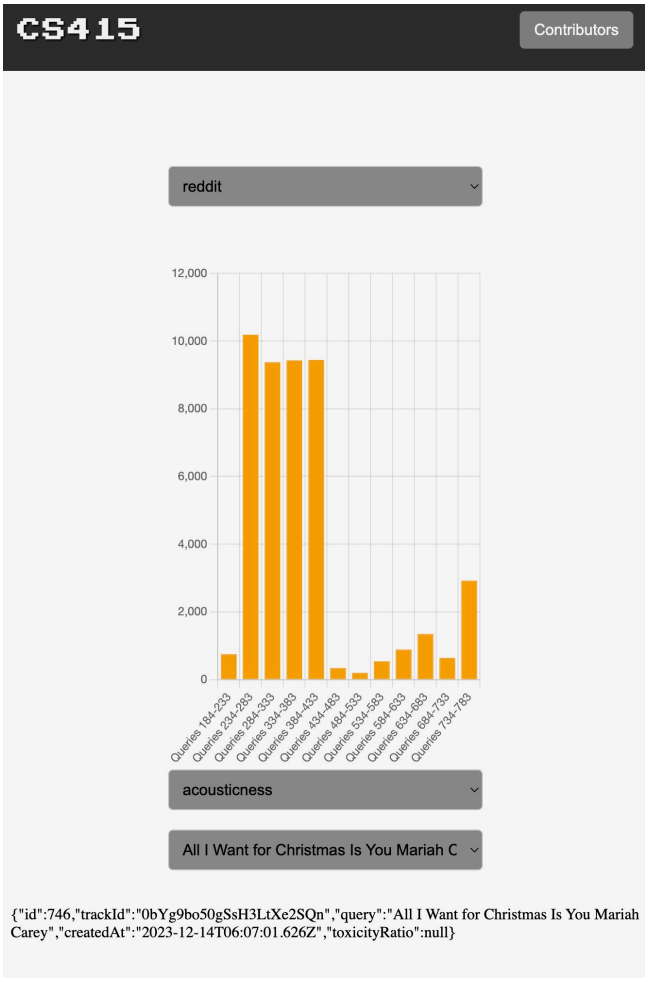

Figure 5: React Application

9 WHAT WE LEARNED

(1) We learned the importance of simplifying our database. In our Prisma model we had a lot of foreign keys and constraints. This became an issue as we did not control the data that was given to us from the different API endpoints of Reddit and Youtube. We had situations where the database could not handle adding more information because a constraint was not satisfied. For example when we were adding Reddit Comments, the comment would not be added because there was a constraint that the parent ID must exist. However, there are situations where the parent comment would not exist. This would prevent us from collecting the comment because of this constraint we created.
(2) Another lesson we learned for databases was the value of having our Query Id exist in multiple different tables. This would have been helpful in more complex Queries where we went from the Query Table to the list of Reddit Posts to the comments in each Reddit post. This resulted in slower performance and a lot of unnecessary joins. It would have been simpler if we had the Query Id be a column for each comment.
(3) We were able to get a greater understanding of the different aspects regarding hosting a web server, for example we had to implement the back end database and create API endpoints that would be able to modify this database.
(4) Further, we learned how to set up Bull, a queuing system.
(5) If we were to implement this project again, we would plan the technologies more in advance. For example, originally we were doing an express server with Python scripts running in the background that would be run using a queueing system. However since the server was running Node, we would have had to have something that converted objects from Python to JavaScript. This is because JS and Python handle objects differently.
(6) We also learned how to use Docker within a team to ensure that our local machines did not have issues running the project. The majority of us in the beginning did not understand how to set up, run or even how Docker worked. But throughout the project as we gained more experience using Docker, we were able to understand how it works.

10 WHAT OUR GAME PLAN WOULD BE FOR ACHEIVING SUCCESS NEXT TIME:

(1) Consolidation of Services:
  (a) Previous Issue: Networking problems arose due to separate services for crawlers, server, and React application.
  (b) Proposed Change: Merge these services into a single, unified service.
  (c) Solution: Implement a framework like Next.js that supports both a REST API and a client application.
(2) Utilization of Next.js:
  (a) Why Next.js: It's capable of handling the required tasks efficiently.
  (b) Improved API Structure: API endpoints would be better organized under different routes using the Next.js App Router.
  (c) Server-Side Rendering: Next.js allows for server-side rendering of React components, enabling the client to execute server actions more effectively. This would allow us to build a fully interactive front-end application atop our back-end system.
(3) API Standardization:
  (a) Current Problem: Lack of uniform design in API endpoints, leading to functional inconsistencies.
  (b) Approach: Simplify API endpoints to focus on CRUD (Create, Read, Update, Delete) operations, creating a more standardized and efficient structure.
  (c) Benefit: This approach allows for building complex logic on top of simple, efficient database operations.
(4) Database Schema Design:
  (a) Initial Strategy: Minimized database size by excluding certain fields in models, leading to data retrieval challenges.

(b) Issue Encountered: Difficulty in accessing necessary fields like foreign key relations in Reddit post or YouTube video models.
(c) Redesign Strategy: Opt for a flattened hierarchy in model relations, including necessary relations even at the cost of redundancy.
(d) Goal: Enhance database operations' efficiency and ease of data access and analysis.