Pickie App

Team Members: Haniyyah Hamid, Arham Khaishagi, Alexis Whitsitt

Introduction

Our main goal was to resolve a recurring issue in our lives and in the lives of the people we know. We have all been in a situation where we want to watch a movie and have absolutely no idea what to put on. This project resolves this issue. It allows the user to input a genre, a tone, and text or use voice to text to create a prompt, which can be something as simple as a genre or full description of exactly what they want. It then generates movie suggestions that best match what they provided.

Base Model

Initially, we started with an imported large dataset from Kaggle with just movies, their themes, and their genres. We extracted the title, release date, tagline, description, running time, rating, theme, and genre from each of these files and put them into a pandas data frame. This data frame is the basis of the entire project. Here we used Seaborn to visualize the data frame, check for missing information, and delete the movies with issues for the sake of avoiding bias.

Then, we took our pandas data frame with all of our now cleaned information and converted it back into a text file, so it is compatible with LangChain. We used a model that takes in a string of text and produces a vector with the semantic meaning. For our purposes, we are using this vector to calculate similarity. More specifically, our website is going to take in the input the user provides as a query and do a similarity search on the tagged description text file we created from the data. For example, if the query is “a movie about aliens,” the results include Aliens, Mars Attacks!, Killer Klowns from Outer Space, and many more.

Next, we did text classification. This is where we identified problems in the dataset and corrected them. For example, there were genres that less than a 1,000 movies were classified as, so they were combined to form simpler, more common genres. This is to ensure there is a chance that each movie from the data set could potentially be selected. We used the transformers library by Hugging Face for zero shot classification. We combined this with the description of the movie and our predetermined list of genres to calculate how accurate each genre is for that particular movie.

Finally we calculated the semantics of each movie’s description. Essentially, we use transformers and the j-hartmann/emotion-english-distilroberta-base model to classify the description of each of the movies. This results in every movie having a distinct semantic analysis. For example, a movie like Barbie has the highest score in “joy” and the lowest in “fear.” These kinds of classifications are essential when “tone” is part of the input. That wraps up the data processing portion.

Initially for the frontend, we attempted to make a simple grad.io user interface. Although this worked well, it wasn't super appealing for users and considering the user experience is an essential part of developing web applications. Because of this, we pivoted to making the frontend with Flask and JavaScript.

Our current frontend uses JavaScript, CSS, and HTML. We used Flask to connect our backend to the webpage. Essentially, it initializes the web application and receives the query from the request. This request is sent to the backend to get the semantic recommendations. This result is put into a json file and displayed using Flask. We used Whisper by OpenAI to do the voice-to-text functionality.

To display the recommended movie posters and descriptions, we have to preprocess the movie data before the website is live.This is done by pulling the information that we have already cleaned and updated previously and putting it into a LangChain document. The text is then converted into vector representations, or embeddings, that can be easily accessed while the website is running. When you request a recommendation, the semantic recommendation will be generated and the poster and description will be pulled from here.

OpenAI analyzes the user input for the sentiment, genre, mood, and length of their ideal movie. These deductions are displayed to the user and if they do not like one of them, they can select a new one either from a dropdown menu or by typing in their ideal. These values are then sent to the backend where we do a similarity search using the Hugging Face model with the description vectors from the database and the query. Then it takes the tone and genre the user selected into account. Movies that are semantically similar and, if the user specified a genre or tone, have high scores in a specific genre or tone are shown to the user.

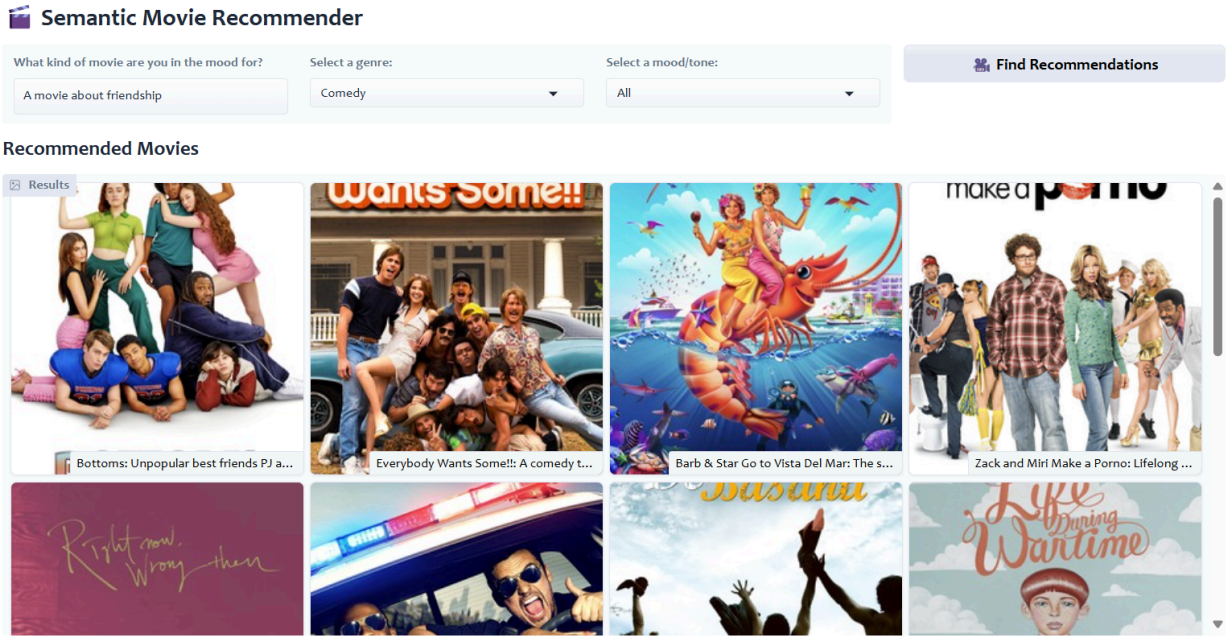
Lessons Learned & Improvements

This was the first time that any of us have worked with any kind of natural language processing library or models, outside of what we have done in class. OpenAI and Flask were both entirely new to us as well. We learned how to take something like a data set and turn it into something unique and customizable. Data sets typically contain static data, so applying natural language processing allowed us to expand the usages for the data sets. We gained so many skills that we ended up developing a full stack web application.

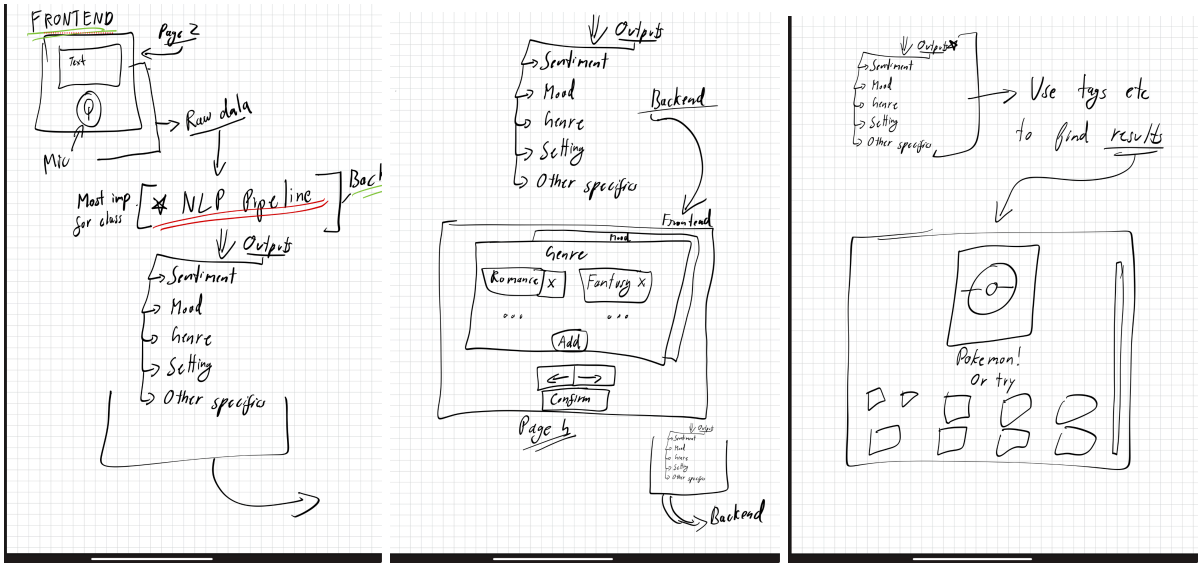
We could potentially expand this project in the future to give the user more options for movie recommendations by using a larger dataset. The current dataset works well for general queries, but if the user is looking for something extremely specific like "pigs," there are not as many results. Another thing we could add is an account system that tracks the user's previous queries, movies they have watched, and movies they want to add to a watchlist. Keeping track of this data would allow us to give them customized recommendations when they aren't looking for something specific. We could also add the option to search based on additional features of the data set, like actors or studios.

Front End Design Process

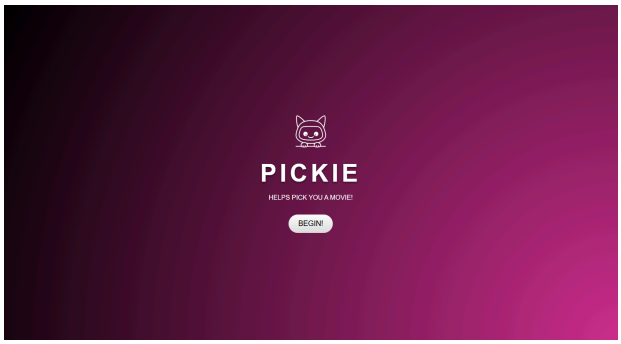
Our initial user interface to test the NLP model's functionality:

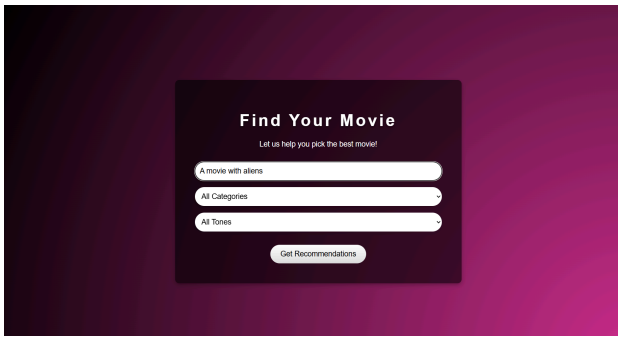


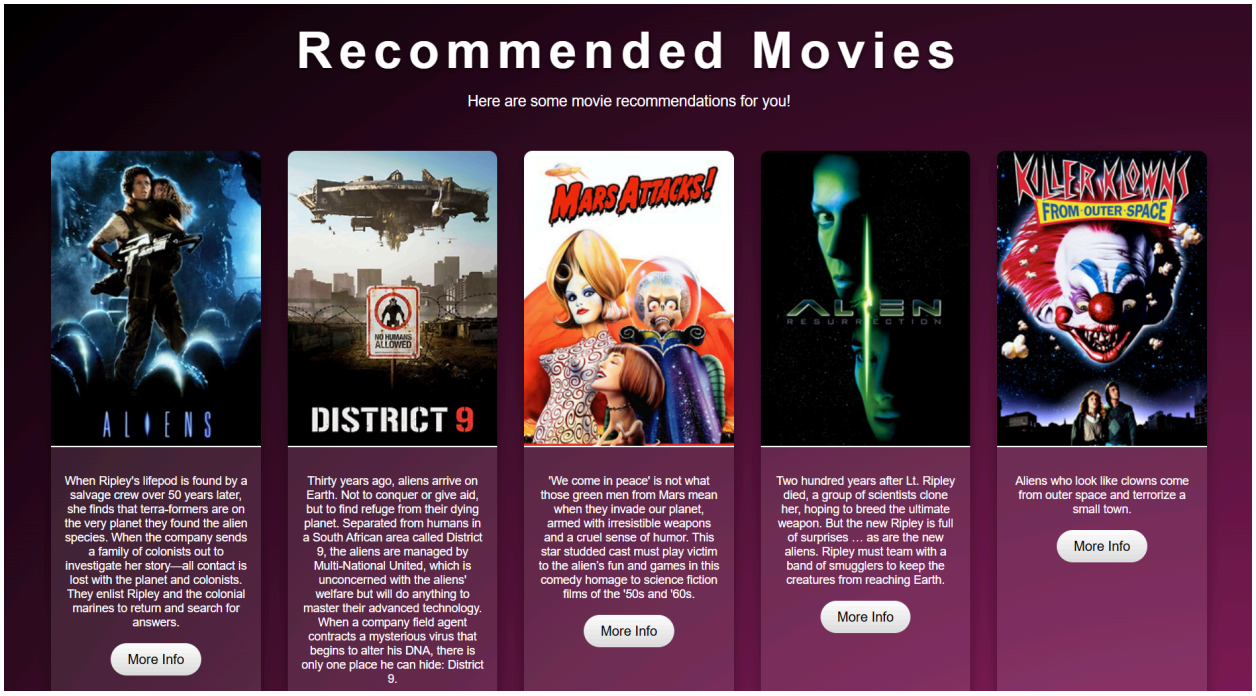
The mockup design for an advanced UI:



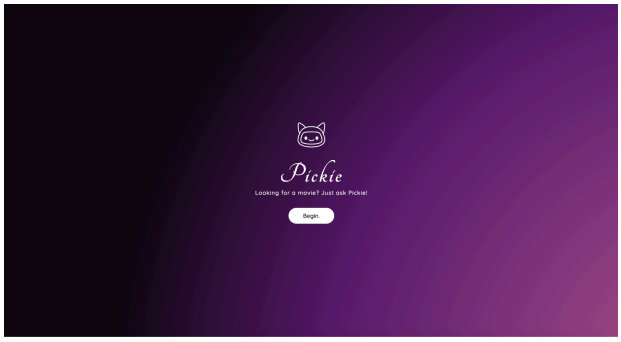
Our first user interface using React JS:

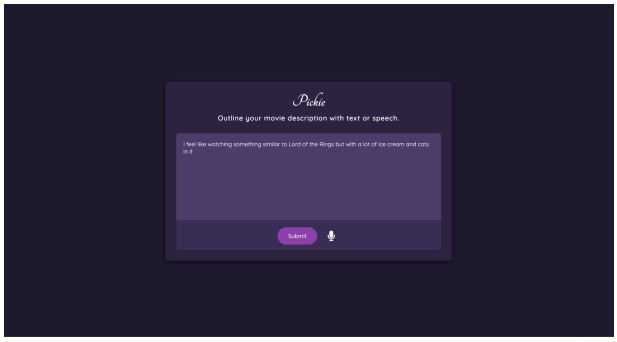


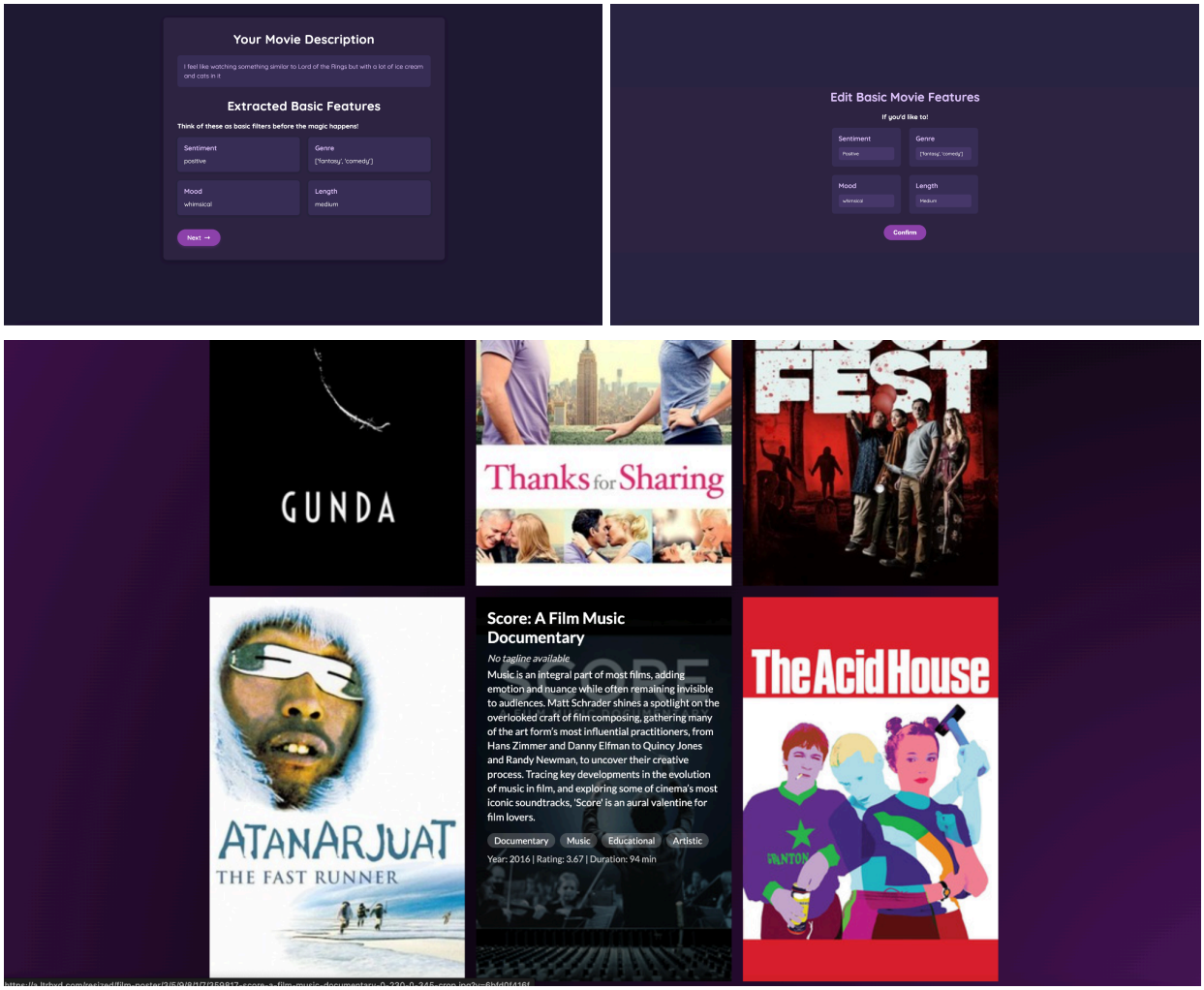




Our final user interface, for ease of connecting APIs and backend logic to frontend:







Testing Outside the Team

We tested this on some of our friends. Alexis, Haniyyah, and Arham got five of their friends to use the webpage. This brought attention to our shortcoming in relation to the specific query “pigs” and similar things. It also showed us that even random letters, such as “sjhdfg,” will also generate movie recommendations. We realized our README file was not entirely accurate through some of our friends attempting to run the file on their personal devices. Although they ran into some of these issues, it did not make the page unusable and they did like the recommendations they were given.

Contributions

Alexis - optimization, debugging, testing, documentation focused

- Wrote the final report
- Assisted in design/visualization/flow design of app
- Data processing

- Data optimization

Haniyyah - primarily backend developer, Flask API, prototyping frontend
- Creation of base model and text classification
- Prototype of base model for visualization
- Basic function React JS UI + Flask API
- Assisted in design/visualization/flow design of app

Arham - primarily frontend developer
- Creation of final Flask frontend
- Implementation of OpenAI for feature extraction, paid for the API key
- Implemented the audio processing input feature with WhisperAPI
- Assisted in design/visualization/flow design of app

Self-Scoring (all bolded are what applies to each member)

Haniyyah Hamid

80 points - significant exploration beyond baseline (couple issues detected and partially solved)

30 points - Innovation or Creativity: Demonstrated unique approaches, such as using novel techniques or creative data augmentations

10 points - highlighted complexity - could be data gathering, error analysis, architecture, optimization etc.
- **Utilizing ChromaDB for persistent vector storage for the backend and speeding up the processing by only running embedding.py once**
- **Text classification (zeroshot) implementation for NaN values in the dataset during data processing**
- **Opting to use Flask frontend over React JS and Gradio**
- **OpenAI integration for voice/audio processing and WhisperAPI**

10 points - discussion of lessons learned and potential improvements

10 points - exceptional visualization/diagrams/repo
- **Assisted group in design of visualizations of the frontend and backend flow**

10 points - discussion of testing outside of the team, on 5 people.
- **Able to get 5 friends to test out the app**

10 points - earned money with the project

Arham Khaishagi

80 points - significant exploration beyond baseline (couple issues detected and partially solved)

30 points - Innovation or Creativity: Demonstrated unique approaches, such as using novel techniques or creative data augmentations

10 points - highlighted complexity - could be data gathering, error analysis, architecture, optimization etc.

- **Utilizing ChromaDB for persistent vector storage for the backend and speeding up the processing by only running embedding.py once**
- **Text classification (zeroshot) implementation for NaN values in the dataset during data processing**
- **Opting to use Flask frontend over React JS and Gradio**
- **OpenAI integration for voice/audio processing and WhisperAPI**

10 points - discussion of lessons learned and potential improvements
10 points - exceptional visualization/diagrams/repo

- **Assisted group in design of visualizations of the frontend and backend flow**

10 points - discussion of testing outside of the team, on 5 people.

- **Able to get 5 friends to test out the app**

Alexis Whitsitt
80 points - significant exploration beyond baseline (couple issues detected and partially solved)
30 points - Innovation or Creativity: Demonstrated unique approaches, such as using novel techniques or creative data augmentations
10 points - highlighted complexity - could be data gathering, error analysis, architecture, optimization etc.

- **Utilizing ChromaDB for persistent vector storage for the backend and speeding up the processing by only running embedding.py once**
- **Text classification (zeroshot) implementation for NaN values in the dataset during data processing**
- **Opting to use Flask frontend over React JS and Gradio**
- **OpenAI integration for voice/audio processing and WhisperAPI**

10 points - discussion of lessons learned and potential improvements
10 points - exceptional visualization/diagrams/repo

- **Assisted group in design of visualizations of the frontend and backend flow**

10 points - discussion of testing outside of the team, on 5 people.

- **Able to get 5 friends to test out the app**

Github Repository

(old) https://github.com/haniyyahh/NLP-Movies-Recommender
(middles)https://github.com/haniyyahh/pickie-app
https://github.com/Elathius/Pickie-flask
(CURRENT AND FINAL!)https://github.com/awhits-tt/pickie-app/tree/main

The first link goes to our original repository, but we ended up making a new one to accommodate for the massive changes we made as well as Git issues. Then we made a third one to combine all of the different workspaces.