

# ChatBot: Recommending New Books

By Rigre Garciandia

# System Description

## Overview

This ChatBot recommends new books to users according to their preferences and a search string. The user preferences are stored as a string of likes and a string of dislikes, and a search string is provided by the user for each recommendation.

## Building the Knowledge Base

To build the knowledge base, I used the following dataset:

<https://www.kaggle.com/datasets/ymaricar/cmu-book-summary-dataset>

## Cleaning the Data

I cleaned up the data from this dataset by removing any entries that did not contain a Genre or Summary. I also cleaned up the summaries by removing any escape characters and any characters that are not letters, as well as making everything lowercase.

## Indexing the Data

To store my knowledge base, I used a **vector database** called weaviate (<https://weaviate.io/>). I used their *text2vec-contextionary* module to vectorize my text data. I chose this vectorizer because it runs fast on a CPU, not requiring a GPU, and still has pretty decent accuracy when querying.

## Querying the Data

In order to give the user relevant recommendations, I need to take into account their preferences (likes and dislikes) as well as the search query (ex: "I want a book with dragons").

In order to do this, I used Weaviate's built-in search API that vectorizes my query. It uses the user query as the concept vector, retrieving vectors in the database that are close to the query (based on the book title, author, genres, and summary). I also leverage the *moveAwayFrom* and *moveTo* parameters of the Weaviate search API to take the user's preferences into account. Here is my code to generate book recommendations:

```
router.post("/query-book", async (req, res, next) => {
  let { search, likes, dislikes } = req.body;

  search = search || "none";
  likes = likes || "none";
  dislikes = dislikes || "none";

  const resText = await client.graphql.get()
    .withClassName('Book')
    .withFields(['bookTitle', 'author', 'genres', 'summary', 'wikipediaArticleId'])
    .withNearText({
      concepts: search,
      moveAwayFrom: {
        concepts: dislikes,
        force: 0.85
      },
      moveTo: {
        concepts: likes,
        force: 0.95
      }
    })
    .withLimit(10)
    .do()

  const nearestBooks = resText.data.Get.Book.map(book => ({
    title: book.bookTitle,
    author: book.author,
    link: `https://en.wikipedia.org/wiki?curid=${book.wikipediaArticleId}`
  }))
  res.json({
    nearestBooks: nearestBooks
  })
})
```

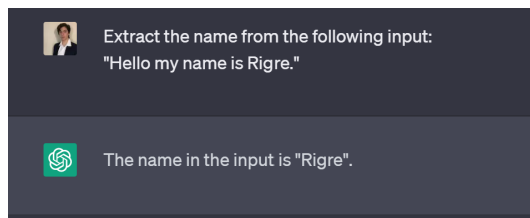
## Parsing User Input and Information Extraction

In order to parse user input, I used OpenAI's *gpt-3.5-turbo* model to parse user input and retrieve information in a standard format. Rather than using a traditional Named Entity Recognition (NET) model like Spacy, I used the OpenAI api because I found it to be more accurate, and the cost was relatively cheap (building and testing this entire project cost me about two cents).

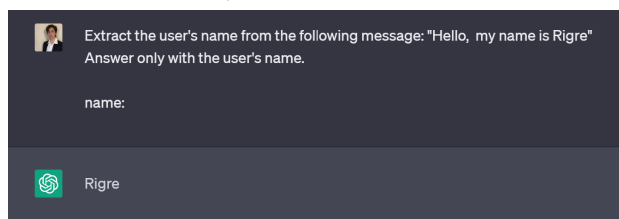
## Engineering the Prompts

In using the OpenAI chat APIs for extracting information, I found that the biggest challenge was getting output in a standard format. The models are not deterministic, and while some of the more advanced models allow you to regulate a *temperature* parameter that makes responses more deterministic, I did not want to spend extra money using these models. My solution was to engineer prompts in a way that would increase the probability of the model giving me the answers in a standard format.

For example, instead of formatting my prompt like this:




I formatted my prompts like this:



The second prompt gives more explicit instructions, and ends with "name:", which leverages the fact that the model is trying to predict what text completes the conversation.


I also found that giving the model examples resulted in more consistent answers. For example:





Can you cleanup the user's preferences to be a comma-separated list of values:  
"I like historical fantasy and romance. I also like when the novel includes political intrigue and shifting power structures"

output only the comma separated list and nothing else. No commas  
For example, for "I dislike romance and comedy, as well as mystery" you output should be:  
List: Romance, comedy, mystery

List:



List: Historical fantasy, romance, political intrigue, shifting power structures



## Extracting the User's Name

```
chatRes = await openai.createChatCompletion({
  model: "gpt-3.5-turbo",
  messages: [{role: "user", content: `Extract the user's name from the following message: "${userInput}"
  Answer only with the user's name.

  User name: `}]
})

let name = chatRes.data.choices[0].message.content;
name = name.toLowerCase()
```

## Extracting the list of preferences in a standard format

```
chatRes = await openai.createChatCompletion({
  model: "gpt-3.5-turbo",
  messages: [{role: "user", content: `Can you cleanup the user's preferences to be a comma-separated list of values:
  "${userInput}"

  output only the comma separated list and nothing else. No commas
  For example, for "I dislike romance and comedy, as well as mystery" you output should be:
  List: Romance, comedy, mystery

  List: `}]
})

const preferences = chatRes.data.choices[0].message.content;
```

## Cleaning Up the User Query for Better Search Results

```
chatRes = await openai.createChatCompletion({
  model: "gpt-3.5-turbo",
  messages: [{role: "user", content: `can you clean the following user input, which is their answer to what they are looking for in a book.
I need the user input to be cleaned up to get the best results when I put it as a query to a vector database containing plot summaries of many different books.
Please output the query and nothing else. Please separate concepts using commas.

Raw user input:
"${userInput}"

Query:`}]
});

const cleanSearch = chatRes.data.choices[0].message.content;
```

## Parsing a Natural Language Response as either Yes or No

```
chatRes = await openai.createChatCompletion({
  model: "gpt-3.5-turbo",
  messages: [{role: "user", content: `can you evaluate the following user input as rather a "Yes" or "No".
Your output should be only one word and nothing else. Please do not include any punctuation.

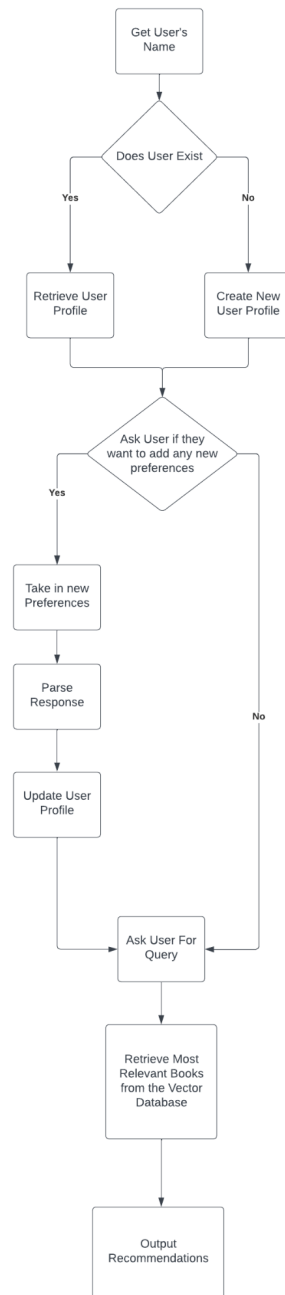
Raw user input:
"${userInput}"

Yes or No:`}]
});
```

## Storing User Profiles

For the purpose of simplicity, I simply store user profiles in a json file that contains their name, likes, and dislikes. Likes and dislikes could be anything like authors, books, or genres. I retrieve user profiles with a simple match on their name. If this project were to be put in production, a more complex user authentication verification mechanism would have to be put in place, and a Database Management System (DBMS) should be used to deal with concurrent edits and further complexities of deployed systems.

# Dialog/Logic Tree



# Sample Dialog Interactions

## First Interaction

```
Assistant: Hello, what is your name?
User: My name is Rigre.
Assistant: Hello Rigre.
Assistant: What are some books, authors, or genres you like?
User: I like historical fantasy novels about knights, lords, and political intrigue. I am also fascinated by Ancient Rome and military history.
Assistant: What are some books, authors, or genres you dislike?
User: I do not like Horror.
Assistant: What kind of book are you interested in reading today?
User: I would like to read a novel about a corrupted society, where the hero starts off as a nobody and ends as king. I also would like it to include dragons and a magical sword.
Assistant: Here are some book recommendations:
1 : Rune War by Joe Dever https://en.wikipedia.org/wiki?curid=5028764
2 : King of Foxes by Raymond E. Feist https://en.wikipedia.org/wiki?curid=31396551
3 : The Tritonian Ring by L. Sprague de Camp https://en.wikipedia.org/wiki?curid=8494835
4 : Conan the Great by Leonard Carpenter https://en.wikipedia.org/wiki?curid=28828819
5 : Farthest Reach by Richard Baker https://en.wikipedia.org/wiki?curid=28287573
6 : The Dragon Waiting: A Masque of History by John M. Ford https://en.wikipedia.org/wiki?curid=12058048
7 : Mydnight's Hero by Joe Dever https://en.wikipedia.org/wiki?curid=5028802
8 : Knight Crusader by Ronald Welch https://en.wikipedia.org/wiki?curid=24095373
9 : The King's Peace by Jo Walton https://en.wikipedia.org/wiki?curid=14984064
10 : King Arthur and His Knights of the Round Table by Roger Lancelyn Green https://en.wikipedia.org/wiki?curid=23510411

Process finished with exit code 0
```

## Resulting User Profile

```
{"rigre":{"likes":" Historical fantasy, knights, lords, political intrigue, Ancient Rome, military history","dislikes":" Horror"}}
```



## Second Interaction

```
Assistant: Hello, what is your name?
User: Hello, I am Rigre
Assistant: Hello Rigre
    According to my records, you like  Historical fantasy, knights, lords, political intrigue, Ancient Rome, military history.
    As far as I know, you dislike  Horror.
    Would you like to add any preferences?
User: I would indeed!
Assistant: What are some books, authors, or genres you like?
User: I like the author Roger Lancelyn Green
Assistant: What are some books, authors, or genres you dislike?
User: I dislike the author Joe Dever
Assistant: What kind of book are you interested in reading today?
User: I would like to read a novel about a corrupted society, where the hero starts off as a nobody and ends up king. I also would like it to include dragons and a magical sword.
Assistant: Here are some book recommendations:
1 : Sir Nobonk and the Terrible Dreadful Awful Naughty Nasty Dragon by Spike Milligan https://en.wikipedia.org/wiki?curid=22159774
2 : Conan the Valiant by Roland J. Green https://en.wikipedia.org/wiki?curid=30757898
3 : The Tritonian Ring by L. Sprague de Camp https://en.wikipedia.org/wiki?curid=8494835
4 : Thorn Ogres of Hagwood by Robin Jarvis https://en.wikipedia.org/wiki?curid=15655813
5 : Conan the Liberator by Lin Carter https://en.wikipedia.org/wiki?curid=6776816
6 : Conan the Hunter by Sean A. Moore https://en.wikipedia.org/wiki?curid=29362596
7 : King Arthur and His Knights of the Round Table by Roger Lancelyn Green https://en.wikipedia.org/wiki?curid=23510411
8 : The Fall of the Templar by Derek Benz https://en.wikipedia.org/wiki?curid=14504641
9 : Forsaken House by Richard Baker https://en.wikipedia.org/wiki?curid=28246359
10 : Moon of the Spider by Richard A. Knaak https://en.wikipedia.org/wiki?curid=20692213

Process finished with exit code 0
```

## Resulting User Profile

```
{"rigre":{"likes":[" Historical fantasy, knights, lords, political intrigue, Ancient Rome, military history Roger Lancelyn Green"],"dislikes":[" Horror Joe Dever"]}}
```

## Observations:

- The phrasing of my answer to the question of whether or not I would like to add preferences does not actually include the words “yes” or “no”
- In the second trial, I added that I dislike the author “Joe Dever”, who was the author of the top suggestion in the first trial. This removed any of this author’s book from the recommended list
- In the second trial, I also added that I liked the author “Roger Lancelyn Green”, which was the last suggestion in the first trial. This resulted in that author’s book being bumped up to position 7 in the second trial

## Appendix A: Knowledge Base Samples

Note: more details as to how the knowledge base was created and queried can be found earlier in this document. The gist of it is that I used this dataset:

<https://www.kaggle.com/datasets/yamaricar/cmu-book-summary-dataset>

I then cleaned up the data (more details above), and put it in a vector database for vector-based search.

Here is a sample entry (clean summary cut off):

```
{"ID":"620","BookTitle":"Animal Farm","Author":"George Orwell","Genres":["Roman \u00e0 clef","Satire","Children's literature","Speculative fiction","Fiction"],"clean_summary":"old major the old boar on the manor farm calls the animals on the farm for a meeting where he compares the humans to parasites and teaches the animals a revolutionary song beasts of england when major dies two young pigs snowball and napoleon assume command and turn his dream into a philosophy the animals revolt and drive the..."}
```

## Appendix B: Sample User Models

```
{"rigre":{"likes":" Historical fantasy, knights, lords, political intrigue, Ancient Rome, military history","dislikes":" Horror"}}
```

```
{"rigre":{"likes":" Historical fantasy, knights, lords, political intrigue, Ancient Rome, military history Roger Lancelyn Green","dislikes":" Horror Joe Dever"}}
```

```
{"karen":{"likes":" George Orwell, political satire","dislikes":"Romance"},"jude":{"likes":" Tolkien","dislikes":" Comedy"}}
```

# Evaluation of the ChatBot

## Strengths

I believe that the vector database does an excellent job for storing and querying relevant results, given that enough quality parameters are given to the query.

I also believe that the use of the OpenAI chat API to parse user input gives the ChatBot more versatility as to the kind of information that it is able to retrieve from user inputs.

## Weaknesses

The dialog flow of the ChatBot is very limited, as it is a fairly linear flow of inputting preferences, a search query, and getting results. The next steps for this project would certainly entail developing a more robust ChatBot that does not fall prey to irregular inputs, as well as making the flow of dialog more dynamic, with the user being able to get as many recommendations as they want, and possibly have an extensive conversation with the ChatBot where the user and the AI assistant work together to figure out the user's likes and dislikes.