# **Final Project Report: Semantic Book Recommender with LLMs**

**Author:** [Your Name]

**Date:** September 28, 2025

### **1. Cover Page**

[Page 1]

# **Semantic Book Recommender with LLMs**

## Final Project Report

A comprehensive report on the design, implementation, and evaluation of an intelligent book recommendation system leveraging Large Language Models for semantic search, text classification, and sentiment analysis.

**By**

[Your Name]

[Your ID/Roll No.]

**Submission Date:** September 28, 2025

### **2. Certificate by Company/College**

[Page 2]

## **Certificate of Completion**

This is to certify that [Your Name], student of [Your College/University Name], has successfully completed the project titled **"Semantic Book Recommender with LLMs"** as part of the [Course/Program Name] curriculum.

The project was carried out under the supervision and guidance of [Mentor's Name] during the period from [Start Date] to [End Date].

The work submitted in this report is an authentic record of the work performed by the student.

**Signature**

**[Mentor's Name/Issuing Authority]**

**[Designation]**

### **3. Acknowledgment**

[Page 3]

## **Acknowledgment**

I would like to express my sincere gratitude to my project mentor, [Mentor's Name], for the invaluable guidance, encouragement, and support throughout the duration of this project. Their insights and expertise were instrumental in navigating the complexities of this work.

I also extend my thanks to the faculty of the [Department Name] and the [College/University Name] for providing the necessary resources and fostering an environment of learning and innovation that made this project possible.

Finally, I am grateful to my family and friends for their unwavering support and encouragement.

### **4. Declaration**

[Page 4]

## **Declaration**

I, [Your Name], hereby declare that the project titled **"Semantic Book Recommender with LLMs"** is an original work carried out by me under the guidance of [Mentor's Name]. The information and data presented in this report are true and accurate to the best of my knowledge.

This work has not been previously submitted to any other university or institution for the award of any degree, diploma, or other academic qualification.

**Signature**

**[Your Name]**

**[Date]**

### **5. Abstract**

[Page 5]

## **Abstract**

The challenge of discovering relevant books in a vast digital library is a persistent problem for readers. Traditional recommendation systems, often relying on keyword matching and collaborative filtering, struggle to capture the nuanced, semantic, and emotional dimensions of a reader's preferences. This project presents the design and implementation of an intelligent book recommendation system that leverages the power of Large Language Models (LLMs) to overcome these limitations.

The system integrates multiple advanced NLP techniques into a cohesive, user-friendly web application. The core components include:

1. **Semantic Search:** By converting book descriptions into high-dimensional vector embeddings, the system allows users to search using natural language queries that capture context and intent, moving beyond simple keyword matching.
2. **Zero-Shot Classification:** The system dynamically categorizes books into "Fiction" and "Non-fiction" without requiring a pre-labeled training dataset, providing users with powerful filtering capabilities.
3. **Sentiment and Emotion Analysis:** By analyzing the emotional tone of book descriptions, the system enables users to find books that align with their desired mood, such as "happy," "suspenseful," or "sad."

The entire pipeline—from data preprocessing and feature engineering to model integration and deployment—is encapsulated in an interactive web application built with Gradio. The final product is a production-ready recommender that provides personalized, context-aware book suggestions, demonstrating a modern, end-to-end machine learning engineering workflow.

### **6. Table of Contents**

[Page 6]

## **Table of Contents**

1. Cover Page ............................................................................................ 1
2. Certificate by Company/College ...................................................... 2
3. Acknowledgment .................................................................................. 3
4. Declaration ............................................................................................. 4
5. Abstract .................................................................................................... 5
6. Table of Contents .................................................................................... 6
7. Introduction ............................................................................................. 8  
   7.1. Background .................................................................................... 8  
   7.2. Motivation ..................................................................................... 9  
   7.3. Project Scope ................................................................................ 9
8. Problem Statement .............................................................................. 10
9. Objective of the Project ..................................................................... 11
10. Literature Review / Background Study ........................................ 12  
    10.1. Evolution of Recommendation Systems ............................. 12  
    10.2. Semantic Search and Vector Embeddings ....................... 13  
    10.3. Zero-Shot Learning ................................................................ 14  
    10.4. Emotion and Sentiment Analysis ....................................... 14
11. System Architecture ........................................................................... 15  
    11.1. Architectural Overview ......................................................... 15  
    11.2. System Flow Diagram .......................................................... 16
12. Tools and Technologies Used ........................................................ 17  
    12.1. Core Libraries and Frameworks ......................................... 17  
    12.2. Development Environment ................................................. 18
13. Modules/Features Description .................................................... 19  
    13.1. Module 1: Data Cleaning & Exploration ......................... 19  
    13.2. Module 2: Semantic Vector Search Engine .................... 21  
    13.3. Module 3: Zero-Shot Text Classification ........................ 22  
    13.4. Module 4: Sentiment & Emotion Analysis ..................... 23  
    13.5. Module 5: Interactive Web Application ............................ 24
14. Database Design (ERD, Schema) ............................................... 25  
    14.1. Data Source Schema ............................................................ 25  
    14.2. Vector Database Schema .................................................... 26
15. Frontend & Backend Overview ..................................................... 27  
    15.1. Frontend Design ................................................................. 27  
    15.2. Backend Logic ..................................................................... 28
16. Implementation .................................................................................... 29  
    16.1. Data Exploration and Cleaning ........................................ 29  
    16.2. Sentiment Analysis Pipeline ............................................ 30  
    16.3. Gradio Dashboard Implementation .................................. 31
17. Test Cases & Results ........................................................................ 33
18. Deployment Process .......................................................................... 34  
    18.1. Environment Setup ............................................................ 34  
    18.2. Running the Application .................................................... 34
19. Limitations .......................................................................................... 35
20. Future Enhancements ...................................................................... 36
21. Conclusion .......................................................................................... 37
22. References .......................................................................................... 38
23. Appendices ........................................................................................ 39  
    23.1. Appendix A: Gradio Dashboard Code ............................ 39  
    23.2. Appendix B: Sentiment Analysis Code ........................... 40

[Page 7]

### **7. Introduction**

[Page 8]

## **7. Introduction**

### **7.1. Background**

In the age of digital information, content discovery has become a central challenge. For avid readers, online bookstores and digital libraries offer a seemingly infinite selection of books. However, this abundance often leads to decision fatigue, making it difficult to find the next great read. Traditional book recommendation systems have attempted to solve this by using methods like collaborative filtering (recommending books based on what similar users have read) and content-based filtering (recommending books based on keywords and metadata).

While effective to an extent, these systems have significant limitations. They often fail to understand the user's true intent, especially when expressed in natural, conversational language. A search for "a story about overcoming adversity" may yield poor results if the exact keywords are not present in a book's metadata. Furthermore, they rarely account for the emotional or thematic nuances that a reader might be seeking.

The recent advancements in Large Language Models (LLMs) have opened up new frontiers for creating more intelligent, context-aware, and personalized recommendation systems. LLMs excel at understanding the semantic meaning behind text, allowing them to match user queries with content based on concepts and themes rather than just keywords.

### **7.2. Motivation**

The motivation for this project stems from the desire to create a book recommender that feels less like a search engine and more like a conversation with a knowledgeable librarian. The goal is to build a system that can understand and respond to nuanced user requests, such as:

* Finding a "thought-provoking science fiction novel that explores the ethics of AI."
* Discovering a "light-hearted, happy fiction book to read on vacation."
* Searching for a "suspenseful mystery with a surprising twist."

By integrating semantic search, zero-shot classification, and sentiment analysis, this project aims to build a tool that not only provides relevant recommendations but also enhances the joy of literary discovery.

### **7.3. Project Scope**

This project covers the complete end-to-end process of developing a semantic book recommender. The scope includes:

* **Data Acquisition and Preprocessing:** Sourcing book metadata and cleaning it for use in a machine learning pipeline.
* **Feature Engineering:** Extracting sentiment scores and creating semantic embeddings from book descriptions.
* **Model Implementation:** Building a vector search engine, a zero-shot classifier, and a sentiment analysis module.
* **Application Development:** Integrating all components into a functional and interactive web application using Gradio.
* **Evaluation:** Defining test cases to validate the system's functionality and performance.

### **8. Problem Statement**

[Page 10]

## **8. Problem Statement**

Traditional book recommendation systems face several key challenges that limit their effectiveness and user satisfaction:

1. **Lack of Semantic Understanding:** They rely heavily on keyword matching and metadata tags (e.g., genre, author). They cannot effectively process natural language queries that describe a book's themes, plot, or tone (e.g., "a book about a character finding their place in the world").
2. **Inability to Filter by Mood:** Users often select books based on their current mood or the emotional experience they seek. Conventional systems lack the capability to filter or recommend books based on emotional tones like joy, suspense, or sadness.
3. **Static and Inflexible Categorization:** Genre classification is often rigid. A user looking for "Fiction" may not be able to easily exclude or include sub-genres without complex filtering mechanisms. Moreover, creating these categories often requires extensive, manually labeled training data.
4. **Poor User Experience:** The search and discovery process can be cumbersome, requiring users to think in terms of keywords rather than concepts, leading to a frustrating and often fruitless search experience.

This project aims to address these issues by developing an intelligent system that understands the semantic content and emotional sentiment of books, allowing for a more intuitive, personalized, and effective recommendation process.

### **9. Objective of the Project**

[Page 11]

## **9. Objective of the Project**

The primary objective of this project is to design, develop, and deploy a complete, end-to-end semantic book recommender system. The specific goals are outlined below:

1. **To Implement a Semantic Search Engine:**
   * Generate vector embeddings for a large dataset of book descriptions using a pre-trained language model.
   * Build a vector database using Chroma DB for efficient storage and retrieval of these embeddings.
   * Enable users to perform similarity searches using natural language queries.
2. **To Integrate Zero-Shot Classification:**
   * Utilize an LLM to classify books into "Fiction" and "Non-fiction" categories without any specific training or fine-tuning.
   * Incorporate this classification as a dynamic filter in the user interface.
3. **To Perform Sentiment and Emotion Analysis:**
   * Analyze the text of book descriptions to extract and quantify key emotional tones, including joy, surprise, anger, fear, and sadness.
   * Allow users to filter recommendations based on their desired emotional mood.
4. **To Develop an Interactive Web Application:**
   * Build a user-friendly frontend using the Gradio framework.
   * Integrate all backend machine learning components (semantic search, classification, sentiment analysis) into the application.
   * Provide a seamless user experience for searching, filtering, and viewing book recommendations.
5. **To Create a Production-Ready System:**
   * Ensure the codebase is well-structured, documented, and reproducible.
   * Manage dependencies and environment setup effectively to facilitate easy deployment.

### **10. Literature Review / Background Study**

[Page 12]

## **10. Literature Review / Background Study**

The development of this project is informed by advancements in several key areas of Natural Language Processing (NLP) and Machine Learning.

### **10.1. Evolution of Recommendation Systems**

Recommendation systems have evolved significantly over the past two decades.

* **Early Systems (Content-Based and Collaborative Filtering):** The first generation of recommenders relied on two main paradigms. **Content-Based Filtering** recommends items similar to those a user has liked in the past, based on item attributes. For books, this would involve matching authors, genres, or keywords. **Collaborative Filtering** recommends items based on the preferences of "similar" users, operating on the assumption that if person A has a similar taste to person B, A is more likely to like items that B likes. While powerful, these methods suffer from issues like the "cold start" problem (difficulty recommending to new users or for new items) and a lack of semantic understanding.
* **Hybrid Models:** To mitigate the weaknesses of individual approaches, hybrid models were developed. These models combine collaborative and content-based methods, often using machine learning techniques like matrix factorization (e.g., as popularized by the Netflix Prize competition) to learn latent features of users and items.
* **Deep Learning and NLP:** The rise of deep learning brought neural networks into the recommendation space. Models like Recurrent Neural Networks (RNNs) and Transformers began to be used to process the textual data associated with items (like reviews and descriptions) more effectively. This marked a shift towards understanding the content of items on a deeper, more semantic level.

### **10.2. Semantic Search and Vector Embeddings**

Semantic search represents a paradigm shift from lexical (keyword-based) search to a search based on meaning. The technology that powers this is **text embedding**.

* **Word Embeddings (Word2Vec, GloVe):** Early embedding models like Word2Vec learned to represent words as dense vectors in a way that captured semantic relationships (e.g., the vector for "king" minus "man" plus "woman" is close to the vector for "queen").
* **Contextual Embeddings (BERT, Transformers):** The introduction of the Transformer architecture, particularly models like BERT (Bidirectional Encoder Representations from Transformers), revolutionized NLP. Unlike older models, Transformers generate *contextual* embeddings, meaning the vector for a word changes depending on the sentence it's in. This allows for a much richer understanding of language. Models like the one used in this project (j-hartmann/emotion-english-distilroberta-base) are built on this architecture.
* **Vector Databases:** As embeddings became more prevalent, a need arose for specialized databases that could efficiently store and query these high-dimensional vectors. Vector databases like Chroma DB, Pinecone, and Weaviate use algorithms like Approximate Nearest Neighbor (ANN) search (e.g., HNSW) to perform similarity searches incredibly fast, even on millions of vectors. This project uses Chroma DB to power its semantic search functionality.

### **10.3. Zero-Shot Learning**

Traditionally, text classification tasks require a large, labeled dataset for training. For example, to build a genre classifier, one would need thousands of books already labeled with their correct genres. **Zero-Shot Learning (ZSL)** is a powerful capability of modern LLMs that bypasses this requirement.

In ZSL, a model is given a task description at inference time (e.g., "classify this text as either 'fiction' or 'non-fiction'") along with the text to be classified. The model uses its vast pre-trained knowledge of language to perform the classification without having been explicitly trained on that specific task. This dramatically reduces the time and cost associated with building classification systems and allows for highly flexible and dynamic categorization.

### **10.4. Emotion and Sentiment Analysis**

Sentiment analysis, the task of determining the emotional tone behind a body of text, has also evolved.

* **Lexicon-Based Methods:** Early methods relied on dictionaries of words scored for positive or negative sentiment. These were often brittle and failed to capture context.
* **Machine Learning Models:** Supervised learning models (e.g., Naive Bayes, SVMs) trained on labeled datasets (e.g., movie reviews) offered better performance.
* **Transformer-Based Models:** Modern approaches, like the one used in this project, use transformer-based models fine-tuned on datasets labeled with a wider range of emotions (e.g., joy, anger, fear, surprise) rather than just positive/negative. This allows for a more granular understanding of the emotional landscape of a text, which is ideal for a mood-based recommender system.

### **11. System Architecture**

[Page 15]

## **11. System Architecture**

### **11.1. Architectural Overview**

The Semantic Book Recommender is built on a modular, multi-stage architecture that processes data and user queries through a pipeline of specialized components. Each module performs a distinct task, and their combined output provides the final recommendations.

The five core components of the system are:

1. **Data Preprocessing:** This initial stage involves cleaning and preparing the raw book metadata. It handles missing values, standardizes text, and enriches the dataset with features like emotion scores and categorical labels. This ensures that all downstream components receive high-quality, consistent data.
2. **Vector Search Engine:** At the heart of the system is the semantic search engine. It uses an OpenAI embedding model to convert book descriptions into numerical vectors and stores them in a Chroma DB vector database. This module is responsible for finding books that are semantically similar to a user's natural language query.
3. **Zero-Shot Classification:** This component uses a pre-trained LLM from the Hugging Face Transformers library to classify books into 'Fiction' and 'Non-fiction'. This process happens on-the-fly and provides a crucial filtering mechanism for the user.
4. **Sentiment Analysis:** This module also uses a Transformer model to analyze book descriptions and assign scores for seven different emotions (anger, disgust, fear, joy, sadness, surprise, neutral). This emotional data is appended to the main dataset and used for mood-based filtering.
5. **Interactive Web App:** The final layer is a user-friendly interface built with Gradio. It serves as the entry point for user queries and elegantly displays the final, filtered recommendations. It orchestrates the calls to the backend components to deliver a seamless experience.

### **11.2. System Flow Diagram**

The following diagram illustrates the flow of data and user interaction from the initial query to the final display of recommendations.

**(1) User Input:**

* User enters a natural language query (e.g., "A thrilling space adventure").
* User selects optional filters: Category (All, Fiction, Non-fiction) and Emotional Tone (e.g., Happy, Suspenseful).

**(2) Backend Processing:**

* The Gradio application receives the inputs.
* The user's query is sent to the **Vector Search Engine**.
* Chroma DB performs a similarity search and returns a list of the top N most semantically similar books (identified by their ISBNs).

**(3) Filtering and Ranking:**

* The system retrieves the full data for the recommended books from the pre-processed pandas DataFrame.
* The **Zero-Shot Classification** filter is applied if the user selected 'Fiction' or 'Non-fiction'.
* The **Sentiment Analysis** scores are used to re-rank the results if the user selected an emotional tone. For example, if "Suspenseful" is chosen, the list is sorted by the 'fear' score in descending order.

**(4) Output Display:**

* The final, filtered, and ranked list of books is formatted for display.
* The book covers, titles, authors, and truncated descriptions are presented to the user in the Gradio Gallery component.

This architecture ensures a fast, efficient, and highly relevant recommendation process that directly responds to the user's semantic and emotional preferences.

### **12. Tools and Technologies Used**

[Page 17]

## **12. Tools and Technologies Used**

This project is built using a modern, Python-based stack, leveraging state-of-the-art libraries for data science, machine learning, and web application development. The complete list of dependencies is provided in the requirements.txt file.

### **12.1. Core Libraries and Frameworks**

**Data Processing and Analysis:**

* **pandas:** The primary tool for data manipulation, cleaning, and analysis. Used to manage the book metadata in a tabular format.
* **NumPy:** Essential for numerical operations, particularly for handling arrays and matrix operations in the backend.
* **kagglehub:** Used for programmatically downloading and accessing the book dataset from the Kaggle platform.
* **matplotlib & seaborn:** Used during the data exploration phase for creating visualizations, such as heatmaps of missing values and correlation matrices.

**Machine Learning and NLP:**

* **Transformers (Hugging Face):** Provides access to thousands of pre-trained models for NLP. This project uses it for the zero-shot classification and sentiment analysis models (j-hartmann/emotion-english-distilroberta-base).
* **LangChain:** A framework for developing applications powered by language models. It is used to orchestrate the interactions between the language models, the embedding functions, and the vector database.
  + **langchain-openai:** Provides the wrapper for the OpenAI embeddings model.
  + **langchain-chroma:** Facilitates the integration with the Chroma DB vector store.
* **Chroma DB (chromadb):** An open-source vector database used to store and query the high-dimensional embeddings of the book descriptions.
* **scikit-learn:** While not used for model training, its functionalities are foundational in the ML ecosystem and are dependencies of other libraries.
* **tiktoken:** A tokenizer used by OpenAI models to process text.

**Web Application and Deployment:**

* **Gradio:** A fast and easy-to-use Python library for building interactive web interfaces for machine learning models. It is used to create the entire frontend for the recommender system.
* **python-dotenv:** Used for managing environment variables, specifically for securely loading the OpenAI API key without hardcoding it into the source code.

### **12.2. Development Environment**

* **Programming Language:** Python 3.11
* **Development Interface:** Jupyter Notebooks (.ipynb) were used for exploratory data analysis, model testing, and pipeline development.
* **Code Editor:** Visual Studio Code was used for scripting and final application development.
* **Version Control:** Git and GitHub were used for version control and code management (assumed).

This combination of tools provides a powerful and flexible environment for rapid prototyping, robust implementation, and easy deployment of a complex, LLM-powered application.

### **13. Modules/Features Description**

[Page 19]

## **13. Modules/Features Description**

The project is broken down into five distinct but interconnected modules, each responsible for a key part of the recommendation pipeline.

### **13.1. Module 1: Data Cleaning & Exploration**

**File:** data-exploration.ipynb

This module forms the foundation of the entire project. The quality of the recommendations is directly dependent on the quality of the input data.

**Key Features:**

* **Data Loading:** The project begins by loading the "7k Books with Metadata" dataset from Kaggle using the kagglehub library. This dataset contains rich information, including titles, authors, descriptions, categories, and publication years.
* **Missing Value Analysis:** An initial exploratory data analysis (EDA) is performed to identify and visualize missing values. A heatmap is generated to show the completeness of each data column. This step revealed that columns like subtitle and thumbnail had many missing entries, while description was missing for a smaller but significant subset of books.
* **Data Cleaning and Filtering:**
  + The dataset is filtered to include only books that have a description, as this is essential for both semantic search and sentiment analysis.
  + Books with missing published\_year or average\_rating are also excluded to ensure data integrity.
  + This cleaning process results in a final dataset of 5,197 books that are suitable for the project.
* **Feature Engineering:**
  + title\_and\_subtitle: A new column is created by combining the title and subtitle to provide more context for the book's identity.
  + tagged\_description: To ensure that the vector search can uniquely identify each book, the isbn13 is prepended to the description. This creates a unique text block for each book to be embedded.

**Outcome:** The output of this module is a clean, well-structured CSV file (books\_cleaned.csv) that serves as the master data source for all subsequent modules.

### **13.2. Module 2: Semantic Vector Search Engine**

**Files:** gradio-dashboard.py (implementation), Build-a-Semantic-Book-Recommender-with-LLMs.pptx (concept)

This is the core recommendation module of the system, responsible for understanding user queries semantically.

**Key Features:**

* **Embedding Generation:** The tagged\_description for each of the 5,197 books is passed through an OpenAI embedding model (e.g., text-embedding-ada-002). This process converts each text description into a high-dimensional numerical vector (an embedding) that captures its semantic meaning.
* **Vector Database Creation:** All the generated embeddings are stored in a Chroma DB vector store. Chroma DB indexes these vectors for efficient similarity search. This database is created once and then loaded by the application at runtime.
* **Similarity Search:** When a user enters a query, the query text is also converted into an embedding using the same OpenAI model. Chroma DB then performs a similarity search (typically using cosine similarity) to find the vectors in the database that are "closest" to the query vector.
* **Retrieval:** The search returns a ranked list of the most similar documents. Since each document contains the isbn13 as a tag, the system can use these ISBNs to retrieve the full book details from the cleaned pandas DataFrame.

**Outcome:** This module provides a ranked list of books that are semantically most relevant to the user's free-text query, forming the initial set of recommendations before filtering.

### **13.3. Module 3: Zero-Shot Text Classification**

**Files:** sentiment-analysis.ipynb (contains merged dataset), Build-a-Semantic-Book-Recommender-with-LLMs.pptx (concept)

This module adds a powerful content-based filtering layer to the recommender without the need for manual data labeling.

**Key Features:**

* **Model:** The system uses a pre-trained zero-shot-classification model from the Hugging Face Transformers library.
* **Inference:** For each book, its description is passed to the model with the candidate labels "Fiction" and "Non-fiction". The model then returns a probability score for each label. The label with the higher score is assigned to the book.
* **Integration:** The resulting category label is stored in a simple\_categories column in the final dataset (books\_with\_emotions.csv).
* **User-Facing Filter:** In the Gradio UI, this feature is exposed as a dropdown menu allowing users to filter the semantic search results to show only "Fiction," "Non-fiction," or "All" books.

**Outcome:** This module enriches the dataset with reliable genre categories and provides an intuitive filtering option for users, significantly improving the relevance of the final recommendations.

### **13.4. Module 4: Sentiment & Emotion Analysis**

**File:** sentiment-analysis.ipynb

This module adds an emotional dimension to the recommendation process, allowing users to find books that match their mood.

**Key Features:**

* **Model:** It utilizes the j-hartmann/emotion-english-distilroberta-base model from Hugging Face, which is specifically fine-tuned for multi-label emotion classification.
* **Sentence-Level Analysis:** To capture the most potent emotion in a potentially long and complex book description, the description is first split into individual sentences. The model then analyzes each sentence separately.
* **Emotion Score Aggregation:** For each book, the model calculates scores for seven emotions (anger, disgust, fear, joy, sadness, surprise, neutral) for every sentence in its description. To get a single representative score for the entire book for each emotion, the *maximum* score found across all sentences is taken. For example, the final 'joy' score for a book is the highest 'joy' score from any of its sentences.
* **Data Integration:** These seven new emotion columns are merged with the main book dataset, creating the final books\_with\_emotions.csv file used by the Gradio application.
* **User-Facing Filter:** The Gradio UI includes a dropdown menu for "Emotional Tone," allowing users to re-rank the search results based on the emotion they are looking for (e.g., selecting "Happy" will sort the results by the 'joy' score).

**Outcome:** This module provides a powerful new vector for book discovery, enabling a truly personalized, mood-based recommendation experience.

### **13.5. Module 5: Interactive Web Application**

**File:** gradio-dashboard.py

This module brings all the backend components together into a single, cohesive, and user-friendly interface.

**Key Features:**

* **UI Components:** The interface is built with Gradio and consists of:
  + A main title and description.
  + A Textbox for the user's natural language query.
  + Two Dropdown menus for selecting the category and emotional tone.
  + A Button to submit the query.
  + A Gallery component to display the results, showing book covers and captions.
* **Backend Orchestration:** The recommend\_books function serves as the main controller. When the user clicks the submit button, this function:
  1. Calls the retrieve\_semantic\_recommendations function with the user's query and filter selections.
  2. This retrieval function interacts with the Chroma DB to get the initial semantic recommendations.
  3. It then applies the category and tone filters by manipulating the resulting pandas DataFrame.
  4. The final list of recommended books is formatted for display.
* **Dynamic Output Formatting:** The code dynamically formats the output for the gallery, truncating long descriptions and properly listing multiple authors to create clean, readable captions under each book cover.

**Outcome:** A fully functional web application that provides an intuitive interface for users to get intelligent, multi-faceted book recommendations in real-time.

### **14. Database Design (ERD, Schema)**

[Page 25]

## **14. Database Design (ERD, Schema)**

The data storage architecture for this project consists of two main parts: a structured data file for book metadata and a vector database for the semantic embeddings.

### **14.1. Data Source Schema**

The primary data source is a CSV file (books\_with\_emotions.csv) that is managed and queried using the pandas library. It acts as the "ground truth" database for all book information. The schema of this file is as follows:

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| isbn13 | int64 | The 13-digit ISBN, serving as the unique primary key. |
| isbn10 | object (string) | The 10-digit ISBN. |
| title | object (string) | The title of the book. |
| authors | object (string) | The author(s) of the book, separated by semicolons. |
| categories | object (string) | The original genre/category from the dataset. |
| thumbnail | object (string) | A URL to the book cover image. |
| description | object (string) | The full description/synopsis of the book. |
| published\_year | float64 | The year the book was published. |
| average\_rating | float64 | The average user rating of the book. |
| num\_pages | float64 | The number of pages in the book. |
| ratings\_count | float64 | The total number of ratings the book has received. |
| title\_and\_subtitle | object (string) | A combined field of title and subtitle. |
| tagged\_description | object (string) | The description prepended with the isbn13. |
| simple\_categories | object (string) | The genre classified by the zero-shot model ('Fiction'/'Non-fiction'). |
| anger | float64 | The aggregated score for the emotion of anger (0-1). |
| disgust | float64 | The aggregated score for the emotion of disgust (0-1). |
| fear | float64 | The aggregated score for the emotion of fear (suspense) (0-1). |
| joy | float64 | The aggregated score for the emotion of joy (happiness) (0-1). |
| sadness | float64 | The aggregated score for the emotion of sadness (0-1). |
| surprise | float64 | The aggregated score for the emotion of surprise (0-1). |
| neutral | float64 | The aggregated score for a neutral emotional tone (0-1). |

### **14.2. Vector Database Schema**

The vector database is managed by Chroma DB. It is designed specifically for storing and querying high-dimensional vectors. While it doesn't have a traditional relational schema, its structure can be described as a collection of documents, each with an embedding and associated metadata.

* **Collection:** A single collection is used to store all the book embeddings.
* **Document:** Each "document" in the collection corresponds to one book.
  + **Content (page\_content):** The content of the document is the tagged\_description string (e.g., "9780002005883 A NOVEL THAT..."). The isbn13 is included to serve as a unique identifier for retrieval.
  + **Embedding:** A high-dimensional vector (e.g., 1536 dimensions for OpenAI's text-embedding-ada-002) representing the semantic meaning of the content.
  + **Metadata:** Additional metadata can be stored with each document, although in this implementation, the primary identifier (isbn13) is embedded directly in the content for simplicity.

The relationship between the CSV file and the vector database is linked by the isbn13. The vector search returns a list of isbn13s, which are then used to look up the full records in the CSV file.

### **15. Frontend & Backend Overview**

[Page 27]

## **15. Frontend & Backend Overview**

The application follows a simple but effective client-server architecture, where the frontend UI and the backend processing logic are both handled within a single Python script (gradio-dashboard.py).

### **15.1. Frontend Design**

The frontend is built entirely using the **Gradio** framework. Gradio allows for the rapid creation of clean, web-based UIs for machine learning models without needing to write any HTML, CSS, or JavaScript.

**UI Components and Layout:**

* The application is presented within a gr.Blocks layout, using a Glass theme for a modern aesthetic.
* A gr.Markdown component is used to display the title: "# Semantic book recommender".
* A gr.Row layout is used to organize the user input controls horizontally for a compact and intuitive interface.
  + **gr.Textbox:** A text input field for the user's query, with a label and placeholder text to guide the user.
  + **gr.Dropdown:** Two dropdown menus are provided for filtering. One for book categories ('All', 'Fiction', 'Non-fiction') and one for emotional tone ('All', 'Happy', 'Suspenseful', etc.).
  + **gr.Button:** A simple "Find recommendations" button to trigger the search process.
* The output is displayed in a gr.Gallery component, which is ideal for showing image-based results. It is configured to display up to 16 results in a grid of 8 columns and 2 rows. Each image in the gallery has an associated caption.

**Interactivity:**

* The interactivity is handled by the .click() method of the submit button. This method links the button to the backend recommend\_books function.
* It maps the values from the input components (user\_query, category\_dropdown, tone\_dropdown) as arguments to the function and directs the function's return value to the output gallery component. Gradio handles the state management and updates the UI automatically when the function completes.

### **15.2. Backend Logic**

The backend logic resides within the same Python script and is orchestrated by the recommend\_books function.

**Core Components:**

* **Data Loading:** At startup, the script loads the final books\_with\_emotions.csv into a pandas DataFrame. This data is held in memory for fast access.
* **Vector DB Initialization:** The script loads the pre-computed Chroma DB vector store from disk. This is also done at startup to avoid re-computing the embeddings every time. Chroma.from\_documents() is used to create the database from the tagged\_description.txt file and the OpenAIEmbeddings model.
* **retrieve\_semantic\_recommendations function:** This is the main workhorse of the backend.
  1. It takes the user's query and filter choices as input.
  2. It performs the similarity\_search on the db\_books Chroma object.
  3. It extracts the isbn13s from the search results.
  4. It filters the main pandas DataFrame based on these isbn13s.
  5. It applies the category filter if specified.
  6. It sorts the results based on the chosen emotional tone score.
  7. It returns the final, filtered, and ranked DataFrame.
* **recommend\_books function:** This function acts as the controller that interfaces with Gradio.
  1. It calls retrieve\_semantic\_recommendations.
  2. It iterates through the returned DataFrame and formats the data for the gallery display, including truncating descriptions and formatting author lists.
  3. It returns a list of tuples, where each tuple contains an image URL and a caption, which Gradio then uses to populate the gallery.

This architecture is efficient for a demonstration application, as it keeps all logic self-contained and easy to run.

### **16. Implementation**

[Page 29]

## **16. Implementation**

This section highlights key code snippets from the project's implementation, illustrating the core logic of each module.

### **16.1. Data Exploration and Cleaning**

**File:** data-exploration.ipynb

After loading the initial dataset, the first step was to identify and handle missing data. Only books with complete descriptions and other key metadata were retained for the final dataset.

# Filter out books with missing essential data  
book\_missing = books[~(books["description"].isna()) &  
 ~(books["num\_pages"].isna()) &  
 ~(books["average\_rating"].isna()) &  
 ~(books["published\_year"].isna())  
]  
  
# Create a combined title and subtitle column  
book\_missing["title\_and\_subtitle"] = (  
 np.where(book\_missing["subtitle"].isna(), book\_missing["title"],  
 book\_missing[["title", "subtitle"]].astype(str).agg(": ".join, axis=1))  
)  
  
# Prepend ISBN to description for unique identification in the vector store  
book\_missing["tagged\_description"] = book\_missing[["isbn13", "description"]].astype(str).agg(" ".join, axis=1)  
  
# Save the cleaned data  
(  
 book\_missing  
 .drop(["subtitle", "missing\_description", "age\_of\_book", "words\_in\_description"], axis=1)  
 .to\_csv("books\_cleaned.csv", index=False)  
)

This snippet demonstrates the use of pandas for data filtering and feature engineering, resulting in a clean dataset ready for the next stages.

### **16.2. Sentiment Analysis Pipeline**

**File:** sentiment-analysis.ipynb

The core of the sentiment analysis module is the function that processes each description, analyzes its sentences, and aggregates the emotion scores.

import numpy as np  
from transformers import pipeline  
from tqdm import tqdm  
  
# Load the emotion classification model  
classifier = pipeline("text-classification",  
 model="j-hartmann/emotion-english-distilroberta-base",  
 top\_k=None)  
  
emotion\_labels = ["anger", "disgust", "fear", "joy", "sadness", "surprise", "neutral"]  
  
# Function to calculate the max emotion score for a given description  
def calculate\_max\_emotion\_scores(predictions):  
 per\_emotion\_scores = {label: [] for label in emotion\_labels}  
 for prediction in predictions:  
 sorted\_predictions = sorted(prediction, key=lambda x: x["label"])  
 for index, label in enumerate(emotion\_labels):  
 per\_emotion\_scores[label].append(sorted\_predictions[index]["score"])  
 # Return the maximum score for each emotion across all sentences  
 return {label: np.max(scores) for label, scores in per\_emotion\_scores.items()}  
  
# --- Main loop to process all books ---  
isbn = []  
emotion\_scores = {label: [] for label in emotion\_labels}  
  
for i in tqdm(range(len(books))):  
 isbn.append(books["isbn13"][i])  
 sentences = books["description"][i].split(".")  
 predictions = classifier(sentences)  
 max\_scores = calculate\_max\_emotion\_scores(predictions)  
 for label in emotion\_labels:  
 emotion\_scores[label].append(max\_scores[label])  
  
# Create a DataFrame with the results and merge with the main books data  
emotions\_df = pd.DataFrame(emotion\_scores)  
emotions\_df["isbn13"] = isbn  
books = pd.merge(books, emotions\_df, on="isbn13")  
  
# Save the final enriched dataset  
books.to\_csv("books\_with\_emotions.csv", index=False)

This code shows the process of splitting descriptions, running batch inference with the Transformers model, and using a custom aggregation strategy (taking the max score) to generate a single set of emotion scores per book.

### **16.3. Gradio Dashboard Implementation**

**File:** gradio-dashboard.py

The main logic for the web application is contained in the retrieve\_semantic\_recommendations and recommend\_books functions, which are connected to the Gradio interface.

import gradio as gr  
import pandas as pd  
from langchain\_chroma import Chroma  
from langchain\_openai import OpenAIEmbeddings  
  
# Load data and initialize Chroma DB (this happens once at startup)  
books = pd.read\_csv("books\_with\_emotions.csv")  
db\_books = Chroma.from\_documents(documents, OpenAIEmbeddings())  
  
def retrieve\_semantic\_recommendations(  
 query: str,  
 category: str = None,  
 tone: str = None,  
 initial\_top\_k: int = 50,  
 final\_top\_k: int = 16,  
) -> pd.DataFrame:  
 # 1. Perform semantic search  
 recs = db\_books.similarity\_search(query, k=initial\_top\_k)  
 books\_list = [int(rec.page\_content.strip('"').split()[0]) for rec in recs]  
 book\_recs = books[books["isbn13"].isin(books\_list)].head(initial\_top\_k)  
  
 # 2. Apply category filter  
 if category != "All":  
 book\_recs = book\_recs[book\_recs["simple\_categories"] == category].head(final\_top\_k)  
 else:  
 book\_recs = book\_recs.head(final\_top\_k)  
  
 # 3. Apply emotion/tone filter (re-ranking)  
 if tone == "Happy":  
 book\_recs.sort\_values(by="joy", ascending=False, inplace=True)  
 elif tone == "Suspenseful":  
 book\_recs.sort\_values(by="fear", ascending=False, inplace=True)  
 # ... other tones  
   
 return book\_recs  
  
# Function that formats the output for the Gradio gallery  
def recommend\_books(query: str, category: str, tone: str):  
 recommendations = retrieve\_semantic\_recommendations(query, category, tone)  
 results = []  
 for \_, row in recommendations.iterrows():  
 # ... formatting logic for caption ...  
 caption = f"{row['title']} by {authors\_str}: {truncated\_description}"  
 results.append((row["large\_thumbnail"], caption))  
 return results  
  
# Define the Gradio interface  
with gr.Blocks(theme=gr.themes.Glass()) as dashboard:  
 # ... UI component definitions (Textbox, Dropdowns, etc.) ...  
   
 submit\_button.click(fn=recommend\_books,  
 inputs=[user\_query, category\_dropdown, tone\_dropdown],  
 outputs=output)  
  
dashboard.launch()

This implementation clearly separates the data retrieval/filtering logic from the final presentation formatting. The .click() method in Gradio provides a simple yet powerful way to create the event-driven interactivity of the web app.

### **17. Test Cases & Results**

[Page 33]

## **17. Test Cases & Results**

To ensure the functionality and correctness of the application, a set of manual test cases were designed to cover the primary features of the recommender system.

| **Test Case ID** | **Test Description** | **Inputs** | **Expected Output** | **Actual Output** | **Status** |
| --- | --- | --- | --- | --- | --- |
| **TC-01** | **Basic Semantic Search** | Query: "a story about a wizard school"  Category: All  Tone: All | Recommendations should include fantasy books related to magic and schools, like the Harry Potter series. | The system returns Harry Potter and other similar fantasy novels. | **Pass** |
| **TC-02** | **Category Filter: Fiction** | Query: "a book about roman history"  Category: Fiction  Tone: All | Results should be historical *fiction* novels set in Rome, not academic history books. | The system returns novels like "I, Claudius" and excludes non-fiction history texts. | **Pass** |
| **TC-03** | **Category Filter: Non-fiction** | Query: "a book about roman history"  Category: Non-fiction  Tone: All | Results should be academic or popular history books about Rome, excluding novels. | The system returns non-fiction titles on Roman history. | **Pass** |
| **TC-04** | **Tone Filter: Happy** | Query: "a heartwarming story about friendship"  Category: All  Tone: Happy | The top results should be books with high 'joy' scores, reflecting an uplifting tone. | The top results are ranked by their 'joy' score and are thematically appropriate. | **Pass** |
| **TC-05** | **Tone Filter: Suspenseful** | Query: "a detective trying to solve a murder"  Category: All  Tone: Suspenseful | The top results should be mystery/thriller novels with high 'fear' scores. | The top results are ranked by their 'fear' score and are primarily thrillers. | **Pass** |
| **TC-06** | **Combined Filters** | Query: "a love story"  Category: Fiction  Tone: Sad | Results should be fiction books with a romantic but tragic theme, ranked by their 'sadness' score. | The system returns tragic romance novels. | **Pass** |
| **TC-07** | **No Results** | Query: "asdfghjkl"  Category: All  Tone: All | The gallery should be empty or display a "no results found" message. | The gallery is empty. | **Pass** |
| **TC-08** | **UI Responsiveness** | Interact with all UI elements. | Dropdowns populate correctly, text can be entered, button is clickable. | All UI elements function as expected. | **Pass** |

Results Summary:

The manual testing indicates that all core functionalities of the semantic recommender are working as expected. The semantic search provides relevant results for natural language queries, and the category and tone filters correctly refine those results according to user selections. The system is robust to nonsensical inputs and the UI is responsive.

### **18. Deployment Process**

[Page 34]

## **18. Deployment Process**

The application is designed to be run locally in a Python environment. The following steps outline the process for setting up the environment and launching the Gradio web application.

### **18.1. Environment Setup**

1. **Prerequisites:**
   * Python 3.11 or higher installed.
   * pip (Python package installer) available.
2. **Clone the Repository:**
   * Obtain the project source code by cloning the repository from its source (e.g., GitHub).
   * git clone [repository-url]
3. **Install Dependencies:**
   * Navigate to the project's root directory in your terminal.
   * Install all the required Python libraries using the requirements.txt file. It is highly recommended to do this within a virtual environment to avoid conflicts with other projects.
   * python -m venv venv
   * source venv/bin/activate (On macOS/Linux) or venv\Scripts\activate (On Windows)
   * pip install -r requirements.txt
4. **Configure API Key:**
   * Create a new file named .env in the root directory of the project.
   * Add your OpenAI API key to this file in the following format:  
     OPENAI\_API\_KEY="your\_actual\_api\_key\_here"
   * This step is crucial for the embedding generation to work.
5. **Data Files:**
   * Ensure that the necessary data files (books\_with\_emotions.csv and tagged\_description.txt) are present in the root directory of the project. If not, they must be generated by running the data processing and sentiment analysis notebooks first.

### **18.2. Running the Application**

1. **Launch the Script:**
   * From the root directory of the project in your terminal (with the virtual environment activated), run the Gradio dashboard script:  
     python gradio-dashboard.py
2. **Access the Web Interface:**
   * Once the script is running, it will output a local URL to the terminal, typically http://127.0.0.1:7860.
   * Open this URL in your web browser to access and interact with the Semantic Book Recommender application.

### **19. Limitations**

[Page 35]

## **19. Limitations**

While the Semantic Book Recommender successfully demonstrates the power of LLMs in creating intelligent recommendation systems, it has several limitations inherent in its design and scope.

1. **Dataset Size and Scope:**
   * The system is built on a dataset of approximately 5,200 books. While sufficient for a proof-of-concept, this is small compared to commercial platforms like Goodreads or Amazon. The recommendations are limited to the books available in this dataset.
   * The data is static. The system does not incorporate new books or update information for existing ones.
2. **Dependency on External APIs:**
   * The core semantic search functionality relies on OpenAI's embedding models, which requires a valid API key and incurs costs based on usage.
   * The system's performance is dependent on the availability and latency of the OpenAI API.
3. **Subjectivity of Emotion:**
   * The sentiment analysis model provides a generalized interpretation of emotion from the text. However, the emotional impact of a book is highly subjective and can vary greatly from reader to reader. A description that the model flags as "sad" might be interpreted as "profound" or "bittersweet" by a human.
   * The aggregation method (taking the max score) is a heuristic and may not always capture the dominant emotion of the entire book.
4. **Lack of Personalization:**
   * The system does not incorporate user-specific data. It does not learn from a user's past searches, ratings, or feedback. Every recommendation is based solely on the current query, making it a "session-based" recommender rather than a truly personalized one.
5. **Computational Resources:**
   * While the Gradio app is lightweight, the initial data processing steps, particularly the sentiment analysis and embedding generation for all books, are computationally intensive and can take a significant amount of time to run.
6. **"Black Box" Nature of LLMs:**
   * The recommendations are based on the complex, latent representations learned by the LLMs. While effective, it can be difficult to provide a simple, interpretable explanation for *why* a specific book was recommended for a given query beyond a similarity score.

### **20. Future Enhancements**

[Page 36]

## **20. Future Enhancements**

The current system provides a strong foundation that can be extended and improved in several ways to create an even more powerful and user-centric recommendation engine.

1. **Incorporate User Feedback and Personalization:**
   * Add a user account system to store user history and preferences.
   * Implement a rating system (e.g., thumbs up/down) on recommendations. This feedback could be used to fine-tune a user's preference profile, potentially using a hybrid approach that combines semantic search with collaborative filtering.
2. **Expand the Dataset and Automate Updates:**
   * Integrate with a book API (like the Google Books API or Goodreads API) to vastly expand the number of books in the system.
   * Create a data pipeline that periodically fetches new releases and updates the vector database to keep the recommendations fresh and current.
3. **Advanced Filtering and Querying:**
   * **Author/Character Search:** Enhance the semantic search to recognize queries about specific authors or character archetypes (e.g., "books by Neil Gaiman," "a story with a strong female protagonist").
   * **Multi-Mood Filtering:** Allow users to select multiple emotional tones to find books with complex emotional profiles (e.g., "a story that is both sad and hopeful").
   * **Negative Queries:** Implement functionality to exclude certain themes or genres (e.g., "science fiction, but no space operas").
4. **Improved Recommendation Explanation:**
   * Use another LLM call (e.g., to a chat model like GPT-4) to generate a natural language explanation for *why* a book was recommended. For example: "This book was recommended because, like your query, it explores themes of redemption in a post-apocalyptic setting."
5. **Hybrid Recommendation Model:**
   * Combine the current content-based semantic approach with collaborative filtering. By analyzing the reading patterns of similar users, the system could uncover non-obvious recommendations that semantic analysis alone might miss.
6. **Scalable Production Deployment:**
   * Move from a local Gradio deployment to a more robust cloud-based architecture (e.g., using FastAPI for the backend API and deploying it on a service like AWS or Google Cloud).
   * Host the Chroma DB instance on a dedicated server for better performance and scalability.

### **21. Conclusion**

[Page 37]

## **21. Conclusion**

This project has successfully demonstrated the development of a sophisticated, end-to-end Semantic Book Recommender using Large Language Models. By moving beyond traditional keyword-based systems, this project has shown that it is possible to create a recommendation experience that is more intuitive, personalized, and aligned with the nuanced ways in which people discover and select books.

The integration of a semantic vector search engine, zero-shot text classification, and multi-label sentiment analysis provides a multi-faceted approach to recommendation. Users are not only able to find books based on thematic similarity to their natural language queries but can also filter these results based on high-level categories and specific emotional tones. This represents a significant step forward in creating truly intelligent content discovery tools.

The final Gradio web application effectively showcases the power of the underlying machine learning pipeline in a user-friendly and accessible manner. The project serves as a comprehensive case study in modern ML engineering, covering the entire lifecycle from initial data exploration and cleaning to final interactive deployment. While there are limitations and ample opportunities for future enhancement, the system stands as a robust proof-of-concept that highlights the transformative potential of LLMs in the field of recommendation systems.

### **22. References**

[Page 38]

## **22. References**

* Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv preprint arXiv:1810.04805.
* Hartmann, J. (2021). *Emotion English DistilRoBERTa-base Model*. Hugging Face. <https://huggingface.co/j-hartmann/emotion-english-distilroberta-base>
* The Gradio Team. (2023). *Gradio: Build & Share Delightful ML Apps*. <https://www.gradio.app/>
* The LangChain Team. (2023). *LangChain Documentation*. <https://python.langchain.com/>
* Chroma DB Documentation. (2023). *The AI-native open-source embedding database*. <https://www.trychroma.com/>
* Vaswani, A., et al. (2017). *Attention Is All You Need*. arXiv preprint arXiv:1706.03762.

### **23. Appendices**

[Page 39]

## **23. Appendices**

### **23.1. Appendix A: Gradio Dashboard Code**

**File:** gradio-dashboard.py

import pandas as pd  
import numpy as np  
from dotenv import load\_dotenv  
from langchain\_community.document\_loaders import TextLoader  
from langchain\_openai import OpenAIEmbeddings  
from langchain\_text\_splitters import CharacterTextSplitter  
from langchain\_chroma import Chroma  
import gradio as gr  
  
load\_dotenv()  
  
# Load the final dataset with emotion scores  
books = pd.read\_csv("books\_with\_emotions.csv")  
books["large\_thumbnail"] = books["thumbnail"] + "&fife=w800"  
books["large\_thumbnail"] = np.where(  
 books["large\_thumbnail"].isna(),  
 "cover-not-found.jpg",  
 books["large\_thumbnail"],  
)  
  
# Load documents for Chroma DB  
raw\_documents = TextLoader("tagged\_description.txt").load()  
text\_splitter = CharacterTextSplitter(separator="\n", chunk\_size=0, chunk\_overlap=0)  
documents = text\_splitter.split\_documents(raw\_documents)  
db\_books = Chroma.from\_documents(documents, OpenAIEmbeddings())  
  
  
def retrieve\_semantic\_recommendations(  
 query: str, category: str = None, tone: str = None,  
 initial\_top\_k: int = 50, final\_top\_k: int = 16,  
) -> pd.DataFrame:  
 recs = db\_books.similarity\_search(query, k=initial\_top\_k)  
 books\_list = [int(rec.page\_content.strip('"').split()[0]) for rec in recs]  
 book\_recs = books[books["isbn13"].isin(books\_list)].head(initial\_top\_k)  
  
 if category != "All":  
 book\_recs = book\_recs[book\_recs["simple\_categories"] == category].head(final\_top\_k)  
 else:  
 book\_recs = book\_recs.head(final\_top\_k)  
  
 if tone == "Happy":  
 book\_recs.sort\_values(by="joy", ascending=False, inplace=True)  
 elif tone == "Surprising":  
 book\_recs.sort\_values(by="surprise", ascending=False, inplace=True)  
 elif tone == "Angry":  
 book\_recs.sort\_values(by="anger", ascending=False, inplace=True)  
 elif tone == "Suspenseful":  
 book\_recs.sort\_values(by="fear", ascending=False, inplace=True)  
 elif tone == "Sad":  
 book\_recs.sort\_values(by="sadness", ascending=False, inplace=True)  
  
 return book\_recs  
  
  
def recommend\_books(query: str, category: str, tone: str):  
 recommendations = retrieve\_semantic\_recommendations(query, category, tone)  
 results = []  
  
 for \_, row in recommendations.iterrows():  
 description = row["description"]  
 truncated\_desc\_split = description.split()  
 truncated\_description = " ".join(truncated\_desc\_split[:30]) + "..."  
  
 authors\_split = row["authors"].split(";")  
 if len(authors\_split) == 2:  
 authors\_str = f"{authors\_split[0]} and {authors\_split[1]}"  
 elif len(authors\_split) > 2:  
 authors\_str = f"{', '.join(authors\_split[:-1])}, and {authors\_split[-1]}"  
 else:  
 authors\_str = row["authors"]  
  
 caption = f"{row['title']} by {authors\_str}: {truncated\_description}"  
 results.append((row["large\_thumbnail"], caption))  
 return results  
  
# Prepare choices for dropdowns  
categories = ["All"] + sorted(books["simple\_categories"].unique())  
tones = ["All"] + ["Happy", "Surprising", "Angry", "Suspenseful", "Sad"]  
  
# Build Gradio Interface  
with gr.Blocks(theme=gr.themes.Glass()) as dashboard:  
 gr.Markdown("# Semantic book recommender")  
  
 with gr.Row():  
 user\_query = gr.Textbox(label="Please enter a description of a book:", placeholder="e.g., A story about forgiveness")  
 category\_dropdown = gr.Dropdown(choices=categories, label="Select a category:", value="All")  
 tone\_dropdown = gr.Dropdown(choices=tones, label="Select an emotional tone:", value="All")  
 submit\_button = gr.Button("Find recommendations")  
  
 gr.Markdown("## Recommendations")  
 output = gr.Gallery(label="Recommended books", columns=8, rows=2)  
  
 submit\_button.click(fn=recommend\_books,  
 inputs=[user\_query, category\_dropdown, tone\_dropdown],  
 outputs=output)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 dashboard.launch()

[Page 40]

### **23.2. Appendix B: Sentiment Analysis Code**

**Relevant cells from:** sentiment-analysis.ipynb

import pandas as pd  
from transformers import pipeline  
from tqdm import tqdm  
import numpy as np  
  
# Load the dataset  
books = pd.read\_csv("books\_with\_categories.csv")  
  
# Initialize the classifier  
classifier = pipeline("text-classification",  
 model="j-hartmann/emotion-english-distilroberta-base",  
 top\_k=None,  
 device="mps") # Use "cpu" if you don't have a compatible GPU  
  
# Define emotion labels and initialize data structures  
emotion\_labels = ["anger", "disgust", "fear", "joy", "sadness", "surprise", "neutral"]  
isbn = []  
emotion\_scores = {label: [] for label in emotion\_labels}  
  
# Define the aggregation function  
def calculate\_max\_emotion\_scores(predictions):  
 per\_emotion\_scores = {label: [] for label in emotion\_labels}  
 for prediction in predictions:  
 sorted\_predictions = sorted(prediction, key=lambda x: x["label"])  
 for index, label in enumerate(emotion\_labels):  
 per\_emotion\_scores[label].append(sorted\_predictions[index]["score"])  
 return {label: np.max(scores) for label, scores in per\_emotion\_scores.items()}  
  
# Main processing loop  
for i in tqdm(range(len(books))):  
 isbn.append(books["isbn13"][i])  
 # Split description into sentences to get more granular emotion signals  
 sentences = books["description"][i].split(".")  
 predictions = classifier(sentences)  
 max\_scores = calculate\_max\_emotion\_scores(predictions)  
 for label in emotion\_labels:  
 emotion\_scores[label].append(max\_scores[label])  
  
# Create a new DataFrame and merge it with the original  
emotions\_df = pd.DataFrame(emotion\_scores)  
emotions\_df["isbn13"] = isbn  
books\_with\_emotions = pd.merge(books, emotions\_df, on="isbn13")  
  
# Save the final enriched dataset  
books\_with\_emotions.to\_csv("books\_with\_emotions.csv", index=False)