Book Recommender System Using LLMs and Vector Search

Semantic Search, LLM-Generated Descriptions, and Interactive UI

Project Documentation

Author & Developer

Mohamed Amine Mammar El Hadj

LinkedIn: www.linkedin.com/in/mohamed-amine-mammar-el-hadj-715a41295
GitHub: https://github.com/mimine47
Email: mohamedamine.devtech@gmail.com

September 2, 2025

A comprehensive approach to building an intelligent book recommendation system leveraging Large Language Models, Vector Embeddings, and Modern UI Technologies

Contents

1 Author Information

1.1 Project Developer

Mohamed Amine Mammar El Hadj
Machine Learning Engineer & Software Developer

- **LinkedIn:** www.linkedin.com/in/mohamed-amine-mammar-el-hadj-715a41295
- **GitHub:** https://github.com/mimine47
- **Email:** mohamedamine.devtech@gmail.com

1.2 Technical Expertise

This project demonstrates proficiency in:

- Large Language Models (LLMs) and Natural Language Processing
- Vector Databases and Semantic Search Technologies
- Machine Learning Pipeline Development
- Full-Stack Application Development
- API Integration and Data Processing
- Modern UI/UX Development with Gradio

2 Project Overview

2.1 Objective

This project aims to build a semantic book recommendation system leveraging:

- Large Language Models (Falcon-7B-Instruct) for **book description generation**
- Vector embeddings for **semantic similarity search**
- Chroma vector database for **persistent, efficient search**
- Gradio for an **interactive frontend** with thumbnails and metadata

2.2 End Goal

Enable users to enter natural language queries (e.g., *"books about magic"*) and receive highly relevant book recommendations with thumbnails and metadata.

3 Motivation & Importance

Traditional keyword-based book searches fail to capture semantic meaning, leading to suboptimal recommendations. This project addresses several key challenges:

- **Semantic Understanding**: Vector search ensures retrieval based on meaning, not just exact keywords
- **Enhanced Metadata**: LLMs generate engaging descriptions, enriching existing metadata
- **User Experience**: Interactive UI improves user engagement and usability
- **Scalability**: Vector databases provide efficient similarity search at scale

4 Project Timeline & Workflow

Table 1 presents the comprehensive project workflow with completion status.

Table 1: Project Timeline and Task Completion Status

Step	Task	Description	Status
1	Data Cleaning & Preparation	Consolidate books_cleaned.csv and description data	✓
2	Description Generation	Use Falcon-7B-Instruct to generate short descriptions	✓
3	Merge Metadata & Descriptions	Ensure all metadata aligns with generated descriptions	✓
4	Text Chunking	Split descriptions into chunks for embedding	✓
5	Vector Embedding	Encode chunks with all-MiniLM-L6-v2	✓
6	Vector DB Creation	Persist embeddings in Chroma	✓
7	Thumbnail Fetching	Use Google Books API to fetch book covers	✓
8	Search Pipeline Development	Combine vector search with metadata matching	✓
9	Frontend Interface	Build Gradio gallery UI for interactive queries	✓
10	Testing & Validation	Evaluate search results & refine pipeline	✓

5 Dataset Details

5.1 Source Files

- `books_cleaned.csv` – book metadata
- `books_with_descriptions_cleaned_new.csv` – LLM-generated descriptions

5.2 Metadata Schema

Table 2 describes the complete metadata schema used in the system.

Table 2: Book Metadata Schema

Column	Description
bookID	Internal unique identifier
title	Book title
authors	Author(s)
average_rating	Average user rating
isbn	ISBN-10
isbn13	ISBN-13
language_code	Language code
num_pages	Page count
ratings_count	Number of ratings
text_reviews_count	Number of reviews
publication_date	Date of publication
publisher	Publisher
isbn13_title	Concatenation of ISBN13 + title
description	LLM-generated short description
thumbnail	URL of book cover

6 Implementation Details

6.1 Book Description Generation

The description generation process utilizes the Falcon-7B-Instruct model to create engaging book descriptions.

```
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
import pandas as pd

# Load dataset and prepare book list
books = pd.read_csv("books_cleaned.csv")
books_list = books["isbn13_title"].tolist()[:2000]

# Initialize Falcon-7B-Instruct model
model_name = "tiiuae/falcon-7b-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    device_map="auto",
    torch_dtype=torch.bfloat16
)

pipe = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    device_map="auto"
)

```

```python
def generate_book_description(book_info, max_new_tokens=100):
    prompt = f"Write a short, engaging description for the book: {book_info}"
    sequences = pipe(
        prompt,
        max_new_tokens=max_new_tokens,
        do_sample=True,
        top_k=10,
        num_return_sequences=1,
        pad_token_id=tokenizer.eos_token_id
    )
    return sequences[0]["generated_text"]

# Batch processing for memory optimization
results = []
batch_size = 500
for start in range(0, len(books_list), batch_size):
    batch = books_list[start:start+batch_size]
    for book in batch:
        desc_text = generate_book_description(book, max_new_tokens=100)
        results.append({
            "isbn13_title": book,
            "description": desc_text
        })

# Save results
desc_df = pd.DataFrame(results)
desc_df.to_csv("books_with_descriptions.csv", index=False)
```

Listing 1: Book Description Generation Pipeline

Key Design Decisions:

- Batch processing optimizes GPU memory usage
- Sampling with `top_k=10` ensures diversity in descriptions
- Maximum token limit prevents overly long descriptions

6.2 Metadata Merging

The metadata merging process combines original book data with generated descriptions.

```python
books = pd.read_csv("books_cleaned.csv")
descriptions = pd.read_csv("books_with_descriptions.csv")

# Merge datasets on common key
final_books_df = books.merge(descriptions, on="isbn13_title", how="inner")
final_books_df.to_csv("final_books_df.csv", index=False)
```

Listing 2: Metadata Merging Process

6.3 Text Chunking for Embeddings

Text chunking optimizes embedding quality by ensuring appropriate context length.

```
from langchain.text_splitter import RecursiveCharacterTextSplitter

splitter = RecursiveCharacterTextSplitter(
    chunk_size=200,
    chunk_overlap=30
)

docs = []
for isbn, desc in zip(final_books_df["isbn13"], final_books_df["
    description"]):
    chunks = splitter.split_text(desc)
    for i, chunk in enumerate(chunks):
        docs.append({
            "isbn13": isbn,
            "chunk_id": i,
            "text": chunk
        })
```

Listing 3: Text Chunking Implementation

Chunking Analysis:

- Most books generate 3–4 chunks
- Maximum observed chunks: 6 (rare occurrences)
- Overlap ensures context preservation across boundaries

6.4 Vector Embedding & Database

The vector database implementation uses Chroma for persistent storage and fast similarity search.

```
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores import Chroma
from langchain.schema import Document

# Initialize embedding model
embedding_fn = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")

# Prepare documents for vector storage
documents = [
    Document(
        page_content=doc["text"],
        metadata={
            "isbn13": doc["isbn13"],
            "chunk_id": doc["chunk_id"]
        }
    ) for doc in docs
]

# Create and persist vector database
vector_db = Chroma.from_documents(
    documents=documents,
    embedding=embedding_fn,
    persist_directory="books_chroma_db"
)
```

```
vector_db.persist()
```

Listing 4: Vector Database Creation

Technology Choices:

- **all-MiniLM-L6-v2**: Fast, high-quality sentence embeddings
- **Chroma**: Provides persistence and efficient similarity search
- **Metadata storage**: Enables full book retrieval from chunk matches

6.5 Thumbnail Fetching

Book cover thumbnails enhance the user interface experience through visual representation.

```python
import requests
import pandas as pd
import time

def fetch_thumbnail(isbn):
    if pd.isna(isbn):
        return "cover-not-found.jpg"

    url = f"https://www.googleapis.com/books/v1/volumes?q=isbn:{isbn}"
    try:
        response = requests.get(url, timeout=10)
        if response.status_code == 200:
            data = response.json()
            return data["items"][0]["volumeInfo"]["imageLinks"]["thumbnail"]
        elif response.status_code == 429:  # Rate limiting
            time.sleep(30)
            return fetch_thumbnail(isbn)
        else:
            return "cover-not-found.jpg"
    except:
        return "cover-not-found.jpg"

# Apply thumbnail fetching
books_df["thumbnail"] = books_df["isbn13"].apply(fetch_thumbnail)
books_df.to_csv("books_with_thumbnails.csv", index=False)
```

Listing 5: Thumbnail Fetching via Google Books API

6.6 Search Pipeline

The search pipeline combines vector similarity search with metadata matching for comprehensive results.

```python
def search_books(query, db, books, k=5):
    """
    Search for books using semantic similarity

    Args:
        query: User search query
        db: Chroma vector database
```

```python
        books: DataFrame containing book metadata
        k: Number of results to return

    Returns:
        DataFrame of matched books with metadata
    """
    results = db.similarity_search(query, k=k)
    matched_books = []

    for r in results:
        isbn = r.metadata.get("isbn13", None)
        if isbn:
            match = books[books["isbn13"] == int(isbn)]
            if not match.empty:
                for _, row in match.iterrows():
                    matched_books.append(row.to_dict())

    return pd.DataFrame(matched_books)

# Example usage
df_matched = search_books("books about love", db, books, k=5)
print(df_matched)
```

Listing 6: Integrated Search Pipeline

6.7 Gradio Frontend Interface

The interactive frontend provides an intuitive gallery-based interface for book discovery.

```python
import gradio as gr

def recommend_books(query):
    """Generate book recommendations based on user query"""
    df = search_books(query, db, books, k=10)
    return [
        (row['thumbnail'], f"{row['title']} by {row['authors']}")
        for _, row in df.iterrows()
    ]

# Create Gradio interface
with gr.Blocks() as demo:
    gr.Markdown("#        Book Recommender System")

    with gr.Row():
        user_query = gr.Textbox(
            label="Enter a description or keyword:",
            placeholder="e.g., 'fantasy novels with magic'"
        )
        submit_btn = gr.Button("Find Books")

    output_gallery = gr.Gallery(
        label="Recommended Books",
        columns=3,
        rows=2
    )

    submit_btn.click(
        fn=recommend_books,
```

```
        inputs=user_query,
        outputs=output_gallery
    )

# Launch application
demo.launch(server_port=7860, server_name="0.0.0.0")
```

Listing 7: Gradio Interactive Interface

7 Key Design Decisions

7.1 Model Selection

- **Falcon-7B-Instruct**: Chosen for human-like description generation capabilities
- **all-MiniLM-L6-v2**: Balance between speed and embedding quality
- **Chroma**: Efficient vector database with persistence support

7.2 Technical Architecture

- **Chunking Strategy**: 200 characters with 30-character overlap optimizes embedding effectiveness
- **Batch Processing**: Memory-efficient description generation
- **API Integration**: Google Books API for enhanced visual experience
- **UI Framework**: Gradio enables rapid deployment with interactive galleries

8 Performance Analysis

8.1 System Metrics

- **Dataset Size**: 2,000 books processed
- **Average Chunks per Book**: 3–4 chunks
- **Embedding Dimensions**: 384 (all-MiniLM-L6-v2)
- **Search Response Time**: $<$ 1 second for typical queries

8.2 Quality Assessment

The system demonstrates strong semantic understanding through several test cases:

- Query: "books about love" → Successfully identifies romance novels
- Query: "fantasy adventure" → Returns fantasy genre books with adventure themes
- Query: "historical fiction" → Accurately matches historical narrative books

9 Future Improvements

9.1 Performance Enhancements

- Implement parallel processing for description generation
- Add caching mechanisms for API calls
- Optimize embedding model selection for domain-specific tasks

9.2 Feature Extensions

- Add filtering capabilities (language, rating, publication date)
- Incorporate user feedback for recommendation refinement
- Implement collaborative filtering alongside content-based recommendations
- Develop mobile-responsive interface

9.3 Scalability Considerations

- Database partitioning for larger datasets
- Distributed embedding computation
- Load balancing for production deployment

10 Conclusion

This project successfully demonstrates the development of a robust, interactive, and intelligent book recommendation system. The integration of Large Language Models, semantic vector search, and modern UI technologies creates a comprehensive solution that addresses the limitations of traditional keyword-based search systems.

The system showcases end-to-end capabilities from raw data processing to deployable frontend application, with detailed documentation of the complete pipeline. The modular architecture ensures maintainability and extensibility for future enhancements.

Key achievements include:

- Successful integration of LLM-generated content with traditional metadata
- Implementation of efficient semantic search using vector embeddings
- Development of an intuitive, visual user interface
- Creation of a scalable, persistent vector database solution

This work provides a foundation for advanced recommendation systems and demonstrates the practical application of modern NLP technologies in information retrieval tasks.

About the Author

Mohamed Amine Mammar El Hadj is a Deep Learning Engineer and Software Developer focusing on NLP, vector search, and AI-powered applications. He has experience building end-to-end deep learning pipelines, including data preprocessing, embedding-based retrieval systems, LLM integration, and deployment-ready interfaces. This book recommender system showcases his practical skills in applying deep learning techniques to real-world problems.

For more projects and technical insights, connect with Mohamed Amine:

- **LinkedIn:** Mohamed Amine Mammar El Hadj
- **GitHub:** github.com/mimine47
- **Email:** mohamedamine.devtech@gmail.com

References

1. Falcon-7B-Instruct Model Documentation, Technology Innovation Institute
2. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks, Reimers & Gurevych, 2019
3. LangChain: Building Applications with LLMs, Harrison Chase et al.
4. Chroma: Open-source embedding database documentation
5. Google Books API Documentation, Google Developers