Synopsis

AI and Machine Learning
4. Semester

Book recommender using NLP

Carsten Lydeking

Zealand Business College

Zealand
Academy of Technologies and Business

Synopsis

AI and Machine Learning
4. Semester

Carsten Lydeking

cal002@edu.zealand.dk

Lecturer, AI and ML:	Jens Peter Andersen
Project Deadline:	30/05/2025
Handed-in:	08/05/25
Semester:	4. Semester
Word Count:	xxx Characters (including spaces)

Cover:	Generated image of binary using DALL-E
Style:	ZBC template – created by Carsten Lydeking (Cally)

Zealand
Academy of Technologies and Business

Contents

1 Introduction

1.1. Motivation

The purpose of this project is to explore the feasibility of running local language models in privacy-preserving environments. To this end, a book recommendation system has been selected as the practical foundation. The recommender leverages Natural Language Processing (NLP) techniques to analyze both book descriptions and free-form user queries, enabling content-based recommendations.

In contrast to most cloud-based recommendation systems, this project demonstrates a fully local setup where all data processing and inference occurs on the user's own machine. This ensures that no user data is transmitted or stored externally, providing strong privacy guarantees.

The goal is to evaluate whether smaller transformer-based models, specifically sentence transformers like MiniLM, are capable of delivering high-quality recommendations in an offline context. Techniques such as semantic embedding, vector similarity search, and metadata filtering are investigated to improve recommendation quality. In doing so, the project explores a broader question of how accessible and effective local AI systems can be for individual users.

—

1.2. Problem Definition

The project is centered around the following research question:

> *How can a local ML model be used to recommend books based on natural language descriptions, relying only on locally running models?*

This leads to three guiding sub-questions:

1. *What techniques exist for embedding text into meaningful vectors?*
2. *How can vector similarity be used for finding similar books?*
3. *What are the limitations of a local, content-only recommender?*

These sub-questions form the conceptual framework for the research and analysis presented in the following chapters.

—

1.3. Planning

The project was conducted over five weeks and divided into the following phases:

- **Week 1 –** Literature review and background research on semantic similarity, embedding models, and recommender systems.
- **Week 2 –** Data exploration and cleaning, including handling of missing values and metadata normalization.
- **Week 3 –** Embedding and indexing: generating dense sentence embeddings and setting up FAISS for similarity search.
- **Week 4 –** Development of a prototype user interface and live experimentation with queries and filters.
- **Week 5 –** Final testing, evaluation, and writing of the synopsis.

2 Methodology and Structure

This project follows a research-based prototype methodology. Rather than developing a commercial software product, the goal is to explore the scientific and technical feasibility of running semantic book recommendations using local machine learning tools. The findings—both theoretical and experimental—constitute the core deliverables of this project.

2.1. Research Approach

The project is based on a combination of literature review and hands-on implementation. Each aspect of the methodology was selected to support answering the sub-questions defined in Section 1.2.

- **Literature Review:** Relevant topics included natural language processing (NLP), sentence embedding techniques, recommender systems, and similarity search. Key technologies such as `MiniLM`, `FAISS`, and `Streamlit` were studied through documentation and other sources. As this field is rapidly evolving, the latest knowledge regarding pracitcal applications was prioritized, e.g. Geron, 2022.
- **Implementation:** The following tools and libraries were used throughout the project:
  - `Python` for scripting, preprocessing, and experimentation.
  - `pandas`, `seaborn`, and `matplotlib` for data analysis and visualization.
  - `sentence-transformers` (MiniLM-L6-v2) to generate semantic vector representations.
  - `FAISS` to index and search high-dimensional embeddings efficiently.
  - `Streamlit` to build an interactive, privacy-preserving user interface.
- **Testing:** Practical testing included a range of user queries and filtering conditions to observe whether recommendations matched the query intent semantically.

—

2.2. Evaluation Criteria

As no supervised training or labeled ground-truth data was involved, the system is evaluated using non-traditional metrics. The following criteria were applied:

- **Qualitative Relevance:** Whether the recommendations appear semantically relevant to a human evaluator.

- **Responsiveness:** How quickly the system responds to user queries on consumer hardware.
- **Offline Capability:** Verification that all processing occurs locally, without internet access.
- **Scalability:** Exploration of performance with larger datasets and indexing sizes.

This methodology supports the central research question posed in section 1.2, and particularly sub-questions 1 and 2, by grounding each system component in both theoretical research and empirical testing.

—

2.3. Structure of the Synopsis

The synopsis is structured with each chapter addressing a specific aspect of the project and each chapter builds upon the previous one to provide an overview of the project.

2.3.1. Main Chapters

- chapter 1 introduces the motivation, problem definition, and research approach.
- chapter 2 describes the research methodology and evaluation criteria.
- chapter 3 explores the dataset used, including its cleaning and preprocessing.
- chapter 4 covers the embedding process, including the theory and implementation of sentence embeddings.
- chapter 5 describes the FAISS indexing and similarity search process.
- chapter 6 describes the user interface design and query workflow.
- chapter 7 discusses the performance evaluation and challenges of the system, including qualitative and quantitative metrics.
- chapter 8 discusses the results, limitations, and future work.

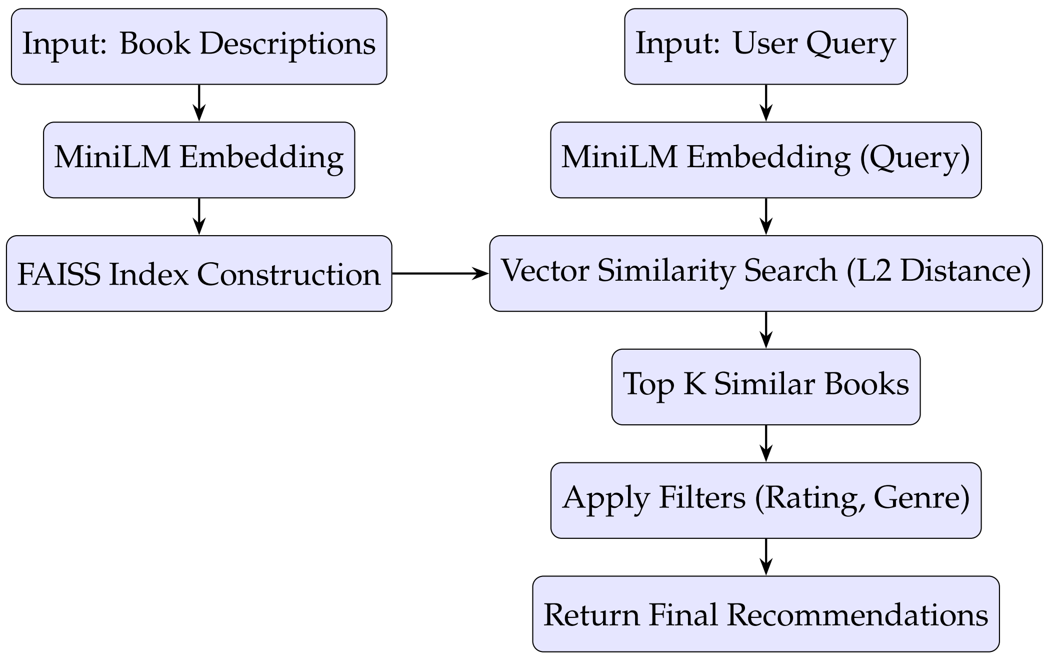
2.3.2. Appendices

A deeper dive into the technical details of the implementation is provided in the appendices, including a more theoretical background on both ML and mathematical concepts. This has been deliberately kept separate from the main chapters, as the focus of the synopsis is on the practical application and results rather than the underlying theory.

—

2.4. Application Architecture

The architecture of the book recommendation system:


Input: Book Descriptions
MiniLM Embedding
FAISS Index Construction
Input: User Query
MiniLM Embedding (Query)
Vector Similarity Search (L2 Distance)
Top K Similar Books
Apply Filters (Rating, Genre)
Return Final Recommendations

3
Dataset Exploration

In order to leverage the dataset Castillo, 2025 used in the recommender system, it has to go through preprocessing steps necessary to enable meaningful analysis. All work was performed using Pythons `pandas`, `numpy`, and `matplotlib/seaborn` libraries, as defined in the data exploration script.

3.1. Dataset Overview

The initial dataset, `books.csv`, contains metadata for 6,810 books. It includes the following fields: `isbn13`, `isbn10`, `title`, `subtitle`, `authors`, `categories`, `thumbnail`, `description`, `published_year`, `average_rating`, `num_pages`, and `ratings_count`.

Some fields, such as `subtitle`, `thumbnail`, and `categories`, had significant missing values. The project focuses on the fields relevant for recommendation: `description`, `average_rating`, `num_pages`, `published_year`, and `authors`.

- `title` — Book title
- `authors` — Author(s)
- `description` — Natural language description (often from publisher blurbs)
- `average_rating` — User rating score
- `num_pages` — Page count
- `published_year` — Year of publication

These attributes were chosen due to their utility in both preprocessing (e.g. filtering) and vector-based analysis.

—

3.2. Cleaning and Filtering

To ensure high-quality input for the NLP model, books with missing or unusable descriptions, ratings, publication years, or page numbers were removed. This resulted in a cleaned dataset, `books_cleaned.csv`, containing 6,507 entries.

Two new features were engineered:

- **missing_description:** Binary indicator showing whether the description was missing.
- **age_of_book:** Computed as 2025 − `published_year`.

—

3.3. Exploratory Analysis

Visual and statistical analysis was performed on the cleaned dataset:

- A **correlation heatmap** (using Spearman correlation) highlights relationships between numerical features such as number of pages, age, and average rating.
- A **distribution plot of average ratings** showed a bell-shaped curve centered around 3.8–4.2. This distribution informed the design of filtering sliders in the UI.
- A **publication year histogram** highlighted a strong skew toward books published after 2000.
- A **top authors chart** visualized the most prolific authors in the dataset.

3.3.1. Correlation Heatmap

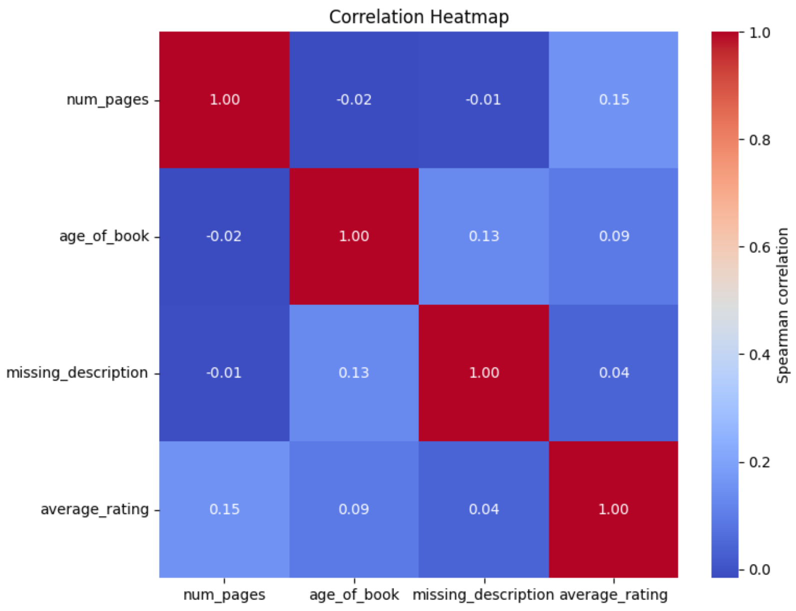


Figure 3.1: Spearman Correlation Heatmap

3.3.2. Rating Distribution

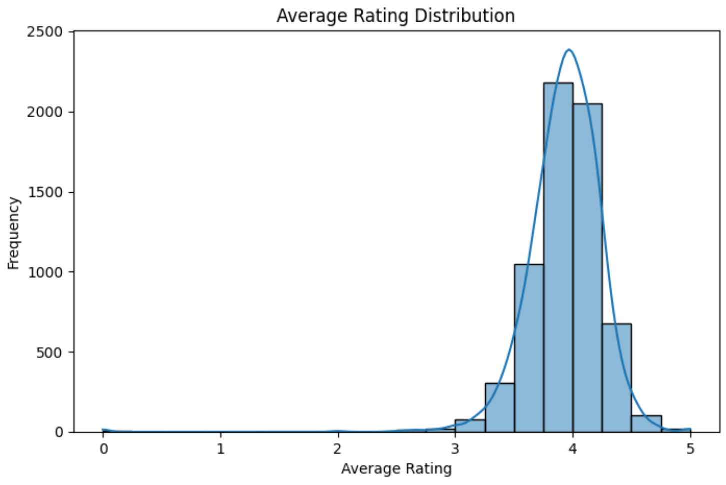


Figure 3.2: Average Rating Distribution

3.3.3. Publication Year Distribution

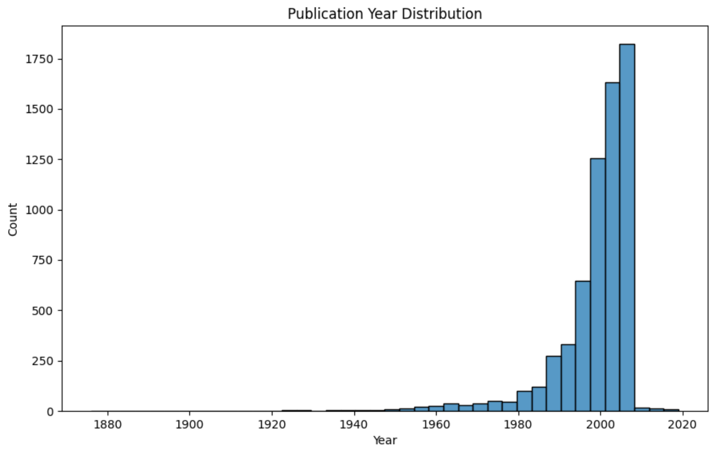


Figure 3.3: Publication Year Distribution

3.3.4. Top Authors

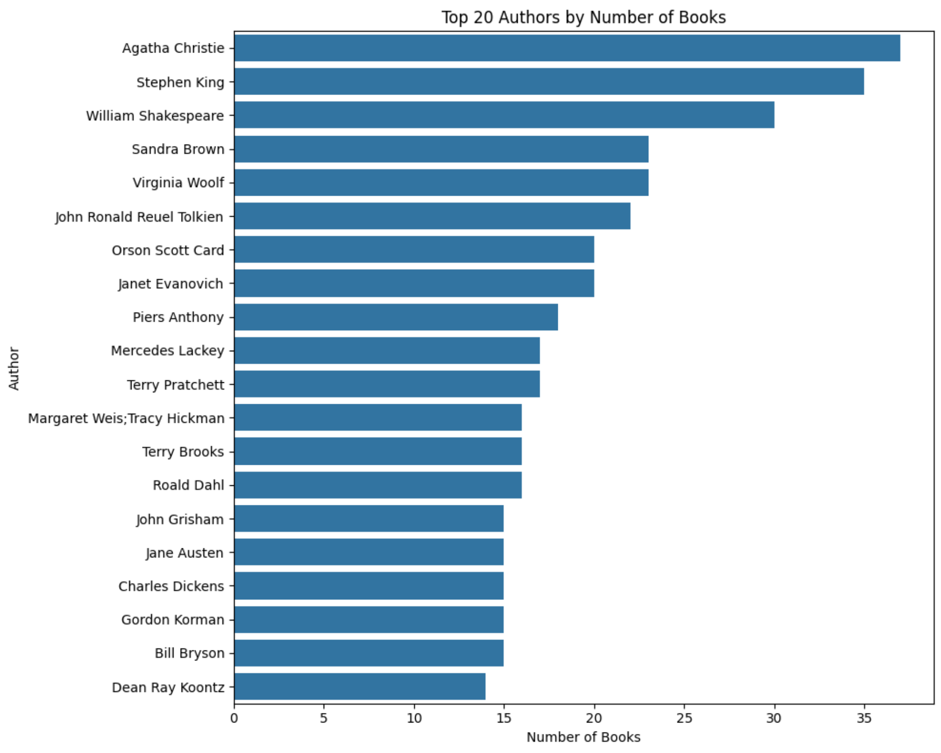


Figure 3.4: Top 20 Authors by Number of Books

These findings confirmed that the dataset was both large and diverse enough to support practical testing of the recommender system. Note that the heatmap in Figure 3.1 shows weak correlation between all members, but the relative strongest between number of pages and the average rating. This may indicate that readers perceive longer books — particularly in fiction — as more substantial, potentially resulting in higher ratings. Also that lower rated books are more rarely published and circulated widely, as seen in the histogram in Figure 3.2.

4
Text Embedding

The embedding took each book description and converted it into a dense vector using a pretrained transformer model, enabling semantic comparison through vector space similarity.

4.1. Sentence Embedding Theory

2

Sentence embeddings are fixed-size numerical vector representations of variable-length text. They preserve semantic meaning and enable similarity search using mathematical distance metrics. The core idea is that similar texts will be embedded close together in high-dimensional space. Formally, for a given input text t, the embedding function f maps it to a vector in $\mathbb{R}^d$:

$$\vec{v} = f(t), \quad \vec{v} \in \mathbb{R}^{384}$$

The model used in this project is the pretrained transformer `all-MiniLM-L6-v2`"sentence-transformers/all-MiniLM-L6-v2", n.d. It outputs 384-dimensional embeddings, which is acceptable for our purpose. This model balances performance with computational efficiency, making it suitable for local execution on consumer hardware.

—

4.2. Embedding Implementation

The cleaned dataset (`books_cleaned.csv`) was used as input. The embedding process involved the following steps:

1. Loading the cleaned book descriptions from the dataset.
2. Converting all descriptions to string type, passing them to the `SentenceTransformer` interface.
3. Encoding the descriptions to obtain a NumPy matrix of embeddings.
4. Storing the resulting matrix in a `FAISS` index for later use in vector similarity search.
5. Saving book metadata in a separate file to allow filtered access to indexed results.

—

4.3. Embedding Configuration

The embedding script uses the following configuration:

- **Model:** `all-MiniLM-L6-v2` from the `sentence-transformers` library.
- **Embedding dimension:** 384
- **Library:** `sentence-transformers` (built on Hugging Face Transformers)
- **Storage:** `FAISS` index with L2 distance
- **Number of embedded books:** 6,507

All embeddings were generated on CPU and saved to the `embeddings/` folder alongside a CSV file (`books_indexed.csv`) containing metadata such as title, author, rating, and description.

—

4.4. Summary

Sentence embeddings serve as the foundation for all downstream recommendation functionality. Using a small yet powerful model such as MiniLM ensures that the system remains performant, even in offline settings. The quality of the embeddings—along with the cleaned, semantically rich descriptions—plays a crucial role in the system's ability to return relevant book suggestions.

5
Vector Similarity Search with FAISS

The system uses vector similarity search to identify semantically related books. Once the text embeddings are generated, the task becomes finding the most similar vectors to a given query vector. This is accomplished using FAISS — Facebook AI Similarity Search“FAISS”, n.d.

5.1. Similarity Search Theory

Given a user query q and a set of embedded book descriptions $\{\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_n\}$, the system computes similarity using the L2 distance:

$$\text{dist}(\vec{q}, \vec{b}_i) = \|\vec{q} - \vec{b}_i\|_2^2$$

Where:

- $\vec{q}$ is the embedding of the user query.
- $\vec{b}_i$ is the embedding of the ith book description.
- The top k results with the smallest distance values are returned as recommendations.

This approach relies on the assumption that semantically similar texts lie close together in the embedding space.

—

5.2. Why FAISS?

FAISS is a library optimized for fast similarity search of dense vectors. It is designed for high-dimensional, large-scale applications and supports both exact and approximate nearest neighbor search.

- **Speed:** Fast search even for large datasets.
- **Flexibility:** Multiple indexing structures and distance metrics.
- **Local Execution:** FAISS supports CPU-only environments and offline usage.

In this project, FAISS was used in its simplest form: `IndexFlatL2`, which performs exact nearest-neighbor search using Euclidean distance.

—

5.3. Implementation

After generating the embeddings, the FAISS pipeline performs the following steps:

1. Create an index using `faiss.IndexFlatL2(dimension)`.
2. Add the matrix of 6,507 book vectors to the index.
3. Save the index to disk as `index.faiss` in the `embeddings/` folder.
4. At query time:
   - The user query is embedded using the same MiniLM model.
   - The query vector is compared against the index to find the top-k nearest neighbors.
   - Metadata for matching books is retrieved from `books_indexed.csv`.

—

5.4. Summary

FAISS allows the system to perform efficient and scalable semantic similarity search over thousands of books. By combining powerful sentence embeddings with fast vector indexing, the recommender system delivers meaningful results in real time—entirely offline.

6
User Interface

While not explicitly part of ML, the design and logic of the user interface (UI) enables interactive querying and filtering of book recommendations. The frontend was built using `Streamlit`, a lightweight Python-based web framework suitable for rapid prototyping and local execution.

6.1. Design Principles

The UI was designed with simplicity, usability, and privacy in mind. All processing, including embedding, similarity search, and filtering, takes place locally. No user data is sent over the internet.

Key goals included:

- Provide an intuitive way for users to search by concept or theme, not keywords.
- Allow flexible filtering based on metadata such as rating or genre.
- Ensure responsiveness without requiring server-side infrastructure.

—

6.2. Query Workflow

The interface presents a single input box where the user can enter a free-form natural language description, such as:

```
“A dystopian society controlled by AI”
```

Once submitted:

1. The query is embedded using the same MiniLM model used for the book descriptions.
2. The query vector is searched against the FAISS index to return the top 50 results.
3. Results are filtered and sorted based on user preferences.

—

6.3. Filtering and Sorting Logic

The UI provides several mechanisms for controlling output:

- **Minimum rating filter:** A slider allows users to set a threshold (e.g., $\geq$ 4.0).
- **Genre/category filter:** A case-insensitive text match on the `categories` field.
- **Sorting:** Results can be sorted by average rating in descending or ascending order.

These filters are applied after similarity search, allowing semantic results to be further refined by user criteria.

—

6.4. Result Display

Each result includes:

- **Book cover image** (fetched from OpenLibrary using ISBN, with fallback).
- **Title and author(s)** in formatted text.
- **Average rating** using a star display.
- **Excerpt of the book description** (up to 500 characters).

Results are displayed in a paginated layout (6 books per page), with navigation controls provided via a page selector widget.

—

6.5. Privacy and Offline Execution

All user inputs and results are processed locally. The system:

- Does not collect or transmit any user data.
- Does not require login, cookies, or tracking.
- Functions without an internet connection (except for optional book cover lookups).

This design supports the overarching motivation for the project: to demonstrate the viability of running compact, privacy-respecting ML models on local hardware.

—

6.6. Summary

The user interface enables real-time, offline interaction with the recommender system. Streamlit provides a simple and effective platform for building and deploying such applications, and the combination of semantic search with filtering yields a user experience that is both flexible and secure.

7
Performance and Evaluation Challenges

This chapter discusses performance evaluation and the challenges involved in applying traditional machine learning metrics to a content-based recommendation system with no user interaction or ground-truth labels.

7.1. Evaluation Without Labels

Unlike supervised learning, the book recommender system does not predict a known output (e.g., a genre, rating, or success score). Instead, it performs semantic search in an embedding space. As a result:

- There is no "correct answer" for what books should be recommended for a given query.
- Standard classification metrics such as accuracy, precision, recall, or F1-score are not applicable.
- User satisfaction, which is often used to measure recommender performance, cannot be captured in an offline and non-interactive system.

7.2. Qualitative Evaluation

The primary evaluation method was qualitative:

- Manually testing a variety of queries to check whether the recommended books aligned with the query intent.
- Verifying that books returned for vague or abstract queries (e.g., "a dystopian society run by algorithms") still showed thematic coherence.
- Testing edge cases such as short queries, factual queries, or genre-specific phrases.

This form of evaluation relies on the developer's or test user's intuition and expectations, and while subjective, it can still provide meaningful insight into whether the system behaves as expected.

7.3. Responsiveness and Latency

Quantitative measurements were more relevant in evaluating system responsiveness:

- **Embedding latency:** MiniLM embedding a single query took < 0.2 seconds on CPU.
- **Similarity search latency:** FAISS returned top-50 neighbors in < 10ms using `IndexFlatL2`.

- **UI response time:** Streamlit rendered results with image loading in under 2 seconds.

All performance metrics were recorded on standard consumer hardware (no GPU), confirming the system's suitability for local deployment.

7.4. Scalability Considerations

While performance was satisfactory for 6,500 books, scalability is still a concern:

- FAISS's exact search is fast for thousands of vectors but may require approximate indexing (e.g., HNSW) for millions.
- The embedding model is fixed. If re-embedding is needed (e.g., after model upgrade), this requires full recomputation.
- Filtering and sorting are done in Python post-query, which may need optimization in larger datasets.

7.5. Limitations of Offline Evaluation

Without user feedback or A/B testing:

- Personal relevance of recommendations is unknown.
- Discovery potential (novel but interesting books) cannot be measured.
- Biases from the dataset (e.g., popular authors, dominant genres) may go unnoticed.

7.6. Summary

Evaluating content-based recommenders in a local, non-interactive setting requires a shift from classical ML metrics to a combination of:

- Qualitative inspection
- Usability testing
- System performance (latency)

If this project were to move further, a future extensions could include a small user study or feedback mechanism to gather more empirical evidence of recommendation quality.

8 Conclusion

8.1. Discussion

The system successfully demonstrates that modern NLP models can be used to recommend books based purely on their textual content, without any need for collaborative data or internet access.

—

8.2. Conclusion

The following sub-questions were addressed:

1. What techniques exist for embedding text into meaningful vectors?
Pretrained transformer-based models like `MiniLM-L6-v2` provide high-quality semantic embeddings. These models output fixed-size vectors that encode context-aware meaning, allowing comparisons based on semantic similarity rather than simple keyword overlap.

2. How can vector similarity be used for finding similar books?
The project utilized L2 distance between embedding vectors to measure similarity. FAISS enabled fast nearest-neighbor retrieval, making it possible to find semantically similar books efficiently even on consumer-grade hardware.

3. What are the limitations of a local, content-only recommender?
The system lacks user personalization and cannot learn from implicit user behavior (e.g., clicks or favorites). Recommendations are purely content-based and may be less diverse. The system also relies heavily on the quality and length of descriptions. However, these trade-offs are acceptable given the privacy and portability benefits of a local-only solution.

Main research question:

> *How can a local ML model be used to recommend books based on natural language descriptions, relying only on locally running models?*

This synopsis has shown that a local recommender system based on sentence embeddings and vector similarity search is not only feasible but also effective. By leveraging compact models like MiniLM and indexing with FAISS, users can interact with an intelligent recommendation engine completely offline.

The findings support the broader hypothesis that smaller LLMs and sentence encoders can power local-first AI systems with practical value, while maintaining user privacy.

—

8.3. Reflection

Throughout the project, several valuable insights were gained:

- **Text preprocessing and quality matter:** The richness of the book descriptions significantly affected the semantic performance of the model.
- **Model performance exceeded expectations:** MiniLM delivered strong semantic matching capabilities despite its small size and fast inference time.
- **Local-first ML is viable:** Embeddings, indexing, and filtering can all be performed within a local Python environment using open-source libraries.
- **Trade-offs exist:** While privacy is enhanced, the lack of personalization and training capabilities limits the scope compared to cloud-based systems.

With more time, additional experiments could be conducted to:

- Compare multiple embedding models (e.g., MPNet, BERT).
- Combine semantic search with metadata-based re-ranking.
- Evaluate using human-labeled query relevance.

Overall, the project provided hands-on experience with modern NLP, vector-based search, and interactive ML applications, combined in a working application.

References

Castillo, D. (2025, May 8). *7k books*. Kaggle. https://www.kaggle.com/datasets/dylanjcastillo/7k-books-with-metadata

Faiss. (n.d.). Retrieved May 8, 2025, from https://faiss.ai/

Geron, A. (2022). *Hands-on machine learning with scikit-learn, keras, and tensorflow: Concepts, tools, and techniques to build intelligent systems* (3nd ed.). O'Reilly Media, Inc. https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/

Sentence-transformers/all-minilm-l6-v2. (n.d.). Retrieved May 8, 2025, from https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

A

Glossary of Mathematical Symbols

This glossary defines the mathematical symbols used throughout the theory and math appendices.

General Symbols

$\mathbb{R}$	The set of real numbers
n	Number of entries (e.g., books or documents)
d	Dimensionality of embedding space (e.g., 384)
i	Index of an individual book or word

—

Vectors and Embeddings

$\vec{v}$	General vector (e.g., sentence embedding)
$\vec{q}$	Query vector (user input)
$\vec{b}_i$	Embedding of the ith book
$\|\vec{v}\|_2$	L2 (Euclidean) norm or length of vector $\vec{v}$
$\vec{a} \cdot \vec{b}$	Dot product between vectors $\vec{a}$ and $\vec{b}$
θ	Angle between two vectors (used in cosine similarity)

—

Matrix Notation

B	Matrix of all book embeddings ($B \in \mathbb{R}^{n \times d}$)
$\vec{q}^T$	Transpose of the query vector (a row vector)
$s = B\vec{q}^T$	Similarity score vector for all books

—

Similarity and Distance

$\|\vec{q} - \vec{b}_i\|_2^2$	Squared Euclidean distance between query and book
$\cos(\theta)$	Cosine similarity between vectors
$\text{softmax}(x)$	Function that converts scores into a probability distribution

—

Self-Attention (Transformers)

Q	Query matrix in self-attention
K	Key matrix
V	Value matrix
QK^T	Raw attention scores between tokens
d_k	Dimension of each key/query vector (used for scaling)
$\text{Attention}(Q, K, V)$	Weighted output of transformer attention

—

Indexes and Search

k	Number of top results to return in nearest-neighbor search
FAISS	Facebook AI Similarity Search — a vector indexing library
`IndexFlatL2`	FAISS index type using exact L2 distance

B
Mathematical Foundations

B.1. Vectors and Vector Spaces

Intuition

A vector is an ordered list of numbers, often used to represent position, direction, or attributes in space. In machine learning, vectors are used to encode things like books, queries, and words.

Example: A 3D vector: $\vec{v} = [2, 1, -3]$ A 384D vector (used in MiniLM): $\vec{v} \in \mathbb{R}^{384}$

Visualization

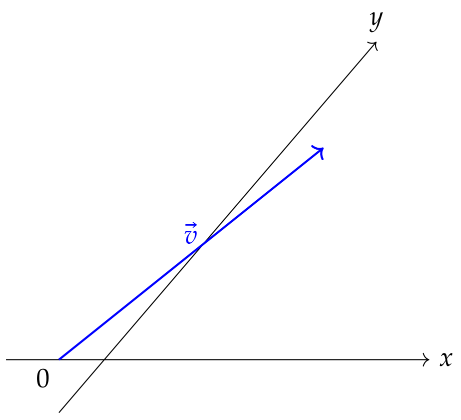


Figure B.1: 2D vector shown as a point in space

Norm and Length

The length of a vector (its L2 norm) is calculated as:

$$\|\vec{v}\|_2 = \sqrt{\sum_{i=1}^{n} v_i^2}$$

This is used in both normalization and distance computations.

—

B.2. Distance and Similarity Measures

L2 (Euclidean) Distance

The L2 distance between two vectors $\vec{a}$ and $\vec{b}$:

$$||\vec{a} - \vec{b}||_2^2 = \sum_{i=1}^{n} (a_i - b_i)^2$$

This is used to compare how similar a user query is to each book.

Cosine Similarity (Alternative)
Cosine similarity measures angle instead of length:

$$\cos(\theta) = \frac{\vec{a} \cdot \vec{b}}{||\vec{a}|| \, ||\vec{b}||}$$

Where $\vec{a} \cdot \vec{b} = \sum a_i b_i$ is the dot product.

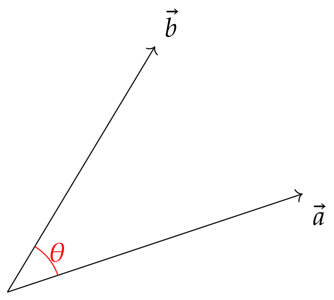


Figure B.2: Angle θ between two vectors (used in cosine similarity)

—

B.3. Matrix Representations and Dot Products

Matrix of Embeddings

When embedding multiple books, we stack vectors into a matrix B:

$$B = \begin{bmatrix} \vec{b}_1 \\ \vec{b}_2 \\ \vdots \\ \vec{b}_n \end{bmatrix} \in \mathbb{R}^{n \times d}$$

Where: - n = number of books (e.g. 6,507) - d = dimensions of each embedding (e.g. 384)

A user query vector $\vec{q} \in \mathbb{R}^d$ can be compared to all rows of B via matrix-vector multiplication.

$$s = B\vec{q}^T$$

This returns a vector of similarity scores $s \in \mathbb{R}^n$.

—

B.4. Self-Attention Scores (MiniLM Core Idea)

Each word in a sentence generates 3 vectors: Query (Q), Key (K), and Value (V). These are used to compute attention weights.

Attention Score Formula

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Where: - QK^T computes raw alignment scores between words - $\sqrt{d_k}$ is a scaling factor - Softmax turns scores into probabilities - V is the weighted sum of context information

I
read
a
book

Figure B.3: Self-attention: word "read" attends to "I", "a", and "book"

—

B.5. Summary of Mathematical Concepts

The math behind this system combines linear algebra, geometry, and probability. The core building blocks are:

- Vectors to represent meaning.
- Distance metrics (L2) for similarity.
- Matrix operations to scale comparisons.
- Self-attention to derive context-aware embeddings.

Together, these allow raw text to be converted into structured representations that machines can reason about efficiently, even in a relatively small local environment.

C
Machine Learning Theory

C.1. Natural Language Processing and Sentence Embeddings

From Words to Vectors

Computers do not understand text in its raw form. To work with natural language, we convert words and sentences into numbers — specifically, into vectors. A vector is a list of numbers that represents some underlying properties or meaning of a word or sentence.

Early methods like Word2Vec or GloVe learned vectors for each word based on the company it keeps (i.e., word context). For example, vectors for `king`, `queen`, and `man` followed patterns like:

$$\texttt{king} - \texttt{man} + \texttt{woman} \approx \texttt{queen}$$

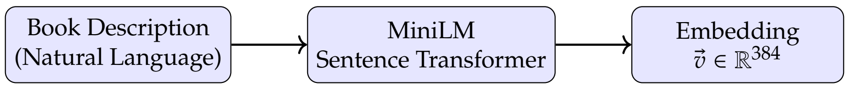
However, word vectors don't consider the meaning of full sentences. That's where *sentence embeddings* come in.

Sentence Embeddings

Sentence embeddings turn an entire sentence or paragraph into a fixed-size vector that captures its meaning. The model used in this project is `all-MiniLM-L6-v2`, which outputs 384-dimensional vectors.

$$\vec{v} = f_{\text{MiniLM}}(t), \quad \vec{v} \in \mathbb{R}^{384}$$

These vectors allow us to compare different pieces of text based on their semantic similarity.



Book Description (Natural Language)
MiniLM Sentence Transformer
Embedding $\vec{v} \in \mathbb{R}^{384}$

Figure C.1: Generating sentence embeddings from raw text

—

C.2. Transformers and Self-Attention

Transformers are a breakthrough architecture in NLP. Instead of reading text one word at a time (like RNNs), they use a mechanism called *self-attention* to process all words simultaneously and understand how each word relates to the others.

Intuition: The Importance of Words

Consider the sentence:

```
"The book she gave me was amazing."
```

The word "she" refers to a person mentioned earlier, and "gave" connects to both "book" and "me". A transformer learns these relationships using self-attention.

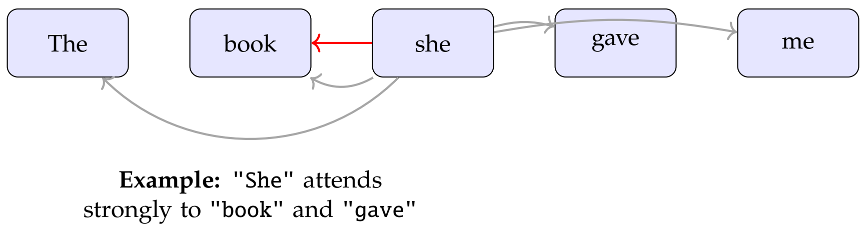


Figure C.2: Simplified self-attention from the word "she" to other words

MiniLM Model

MiniLM is a compressed, fast version of a larger model (BERT). It retains the ability to understand sentence-level semantics while being efficient enough to run locally. It outputs a single vector for each input sentence by aggregating hidden states via mean pooling.

—

C.3. Vector Similarity and Distance Metrics

High-Dimensional Space

Each sentence is embedded into a 384-dimensional vector space. Although we cant visualize this directly, we can think of it as placing each sentence as a point in a "cloud" of meaning. The closer two points are, the more semantically similar they are.

L2 (Euclidean) Distance

To find the most similar book to a user query, we measure the distance between vectors. The most common distance metric is the L2 (Euclidean) distance:

$$\text{dist}(\vec{q}, \vec{b}_i) = \|\vec{q} - \vec{b}_i\|_2^2 = \sum_{j=1}^{384} (q_j - b_{i,j})^2$$

Where:

- $\vec{q}$ is the query embedding.
- $\vec{b}_i$ is the embedding of the ith book.

Alternatively, cosine similarity can be used, which compares the angle between vectors rather than their absolute distance, see Figure C.3.

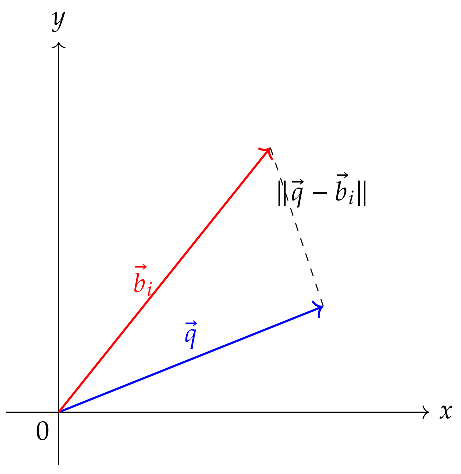


Figure C.3: Semantic similarity as distance between vectors

—

C.4. Nearest Neighbor Search with FAISS

Searching for the closest vectors becomes computationally expensive as the dataset grows. FAISS (Facebook AI Similarity Search) is an open-source library for efficient nearest neighbor search in high-dimensional spaces.

IndexFlatL2

In this project, we use `IndexFlatL2`, which performs exact nearest-neighbor search based on L2 distance. While not the most scalable option for millions of vectors, it is fast and accurate for a few thousand entries.

FAISS indexes the vectors and allows quick lookup of the top-k nearest neighbors to a given query vector.

—

C.5. Recommender Systems: Content-Based Filtering

Two Main Types

There are two main types of recommender systems:

- **Collaborative filtering:** Based on user behavior and preferences.
- **Content-based filtering:** Based on item attributes (e.g., descriptions, genres).

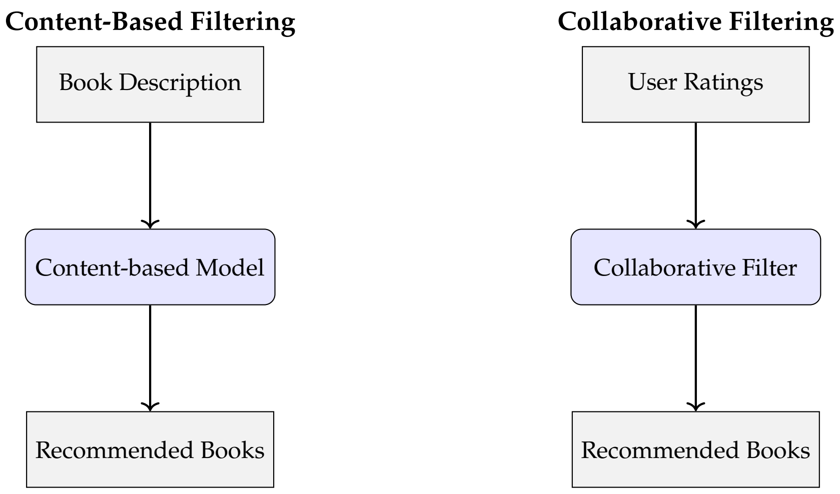


Figure C.4: Filtering approaches: content-based vs. collaborative

Why Content-Based?

In this project, we use content-based filtering because:

- No user data is collected (privacy-by-design).
- Recommendations are based on semantic similarity between descriptions.
- The system runs fully offline.

This approach allows users to describe what they want in natural language and receive books that match the idea, not just a specific keyword.

—

C.6. Local ML vs. Cloud ML

Running ML models locally has several trade-offs:

Advantages

- **Privacy:** No data leaves the user's device.
- **Offline access:** The system works without internet.
- **Cost:** No cloud infrastructure needed.

Limitations

- Limited personalization or learning over time.
- More constrained by local hardware.
- Cannot use large models like GPT-4 or full BERT with ease.

Still, models like MiniLM make local semantic search practical and useful in real-world applications.