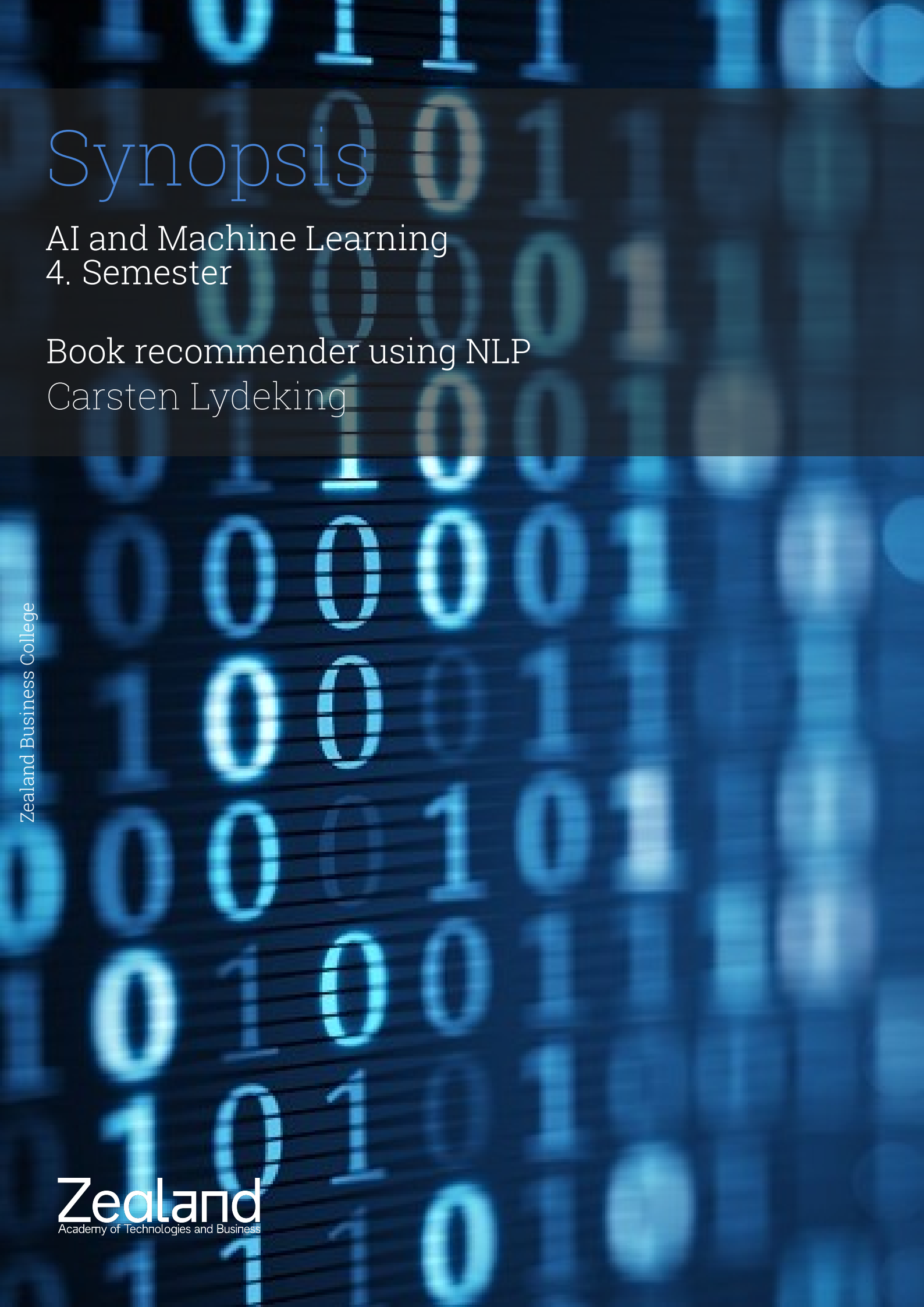


Synopsis

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:	27/05/25
Semester:	4. Semester
Word Count:	22.838 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**
- 1.1 Motivation . . . 1
- 1.2 Problem Definition . . . 1
- 1.3 Link to code repository . . . 1

2 Methodology and Structure **2**
- 2.1 Research Approach . . . 2
- 2.2 Evaluation Criteria . . . 2
- 2.3 Structure of the Synopsis . . . 3
  - 2.3.1 Main Chapters . . . 3
  - 2.3.2 Appendices . . . 3
- 2.4 Application Architecture . . . 3

3 Dataset Exploration **4**
- 3.1 Dataset Overview . . . 4
- 3.2 Data Cleaning and Augmentation . . . 4
- 3.3 Feature Engineering . . . 4
- 3.4 Exploratory Analysis . . . 5
- 3.5 Outcome . . . 5

4 Category Inference **6**
- 4.1 Zero-Shot Classification with BART-MNLI . . . 6
- 4.2 Fallback Keyword Enrichment . . . 6
- 4.3 Refinement, Merging, and Metrics . . . 6
- 4.4 Confidence Analysis and Filtering . . . 6
- 4.5 Length vs Confidence Correlation . . . 7

5 Text Embedding **8**
- 5.1 Sentence Embedding Theory . . . 8
- 5.2 Embedding Implementation . . . 8
- 5.3 Embedding Configuration . . . 8
- 5.4 Summary . . . 9

6 Vector Similarity Search with FAISS **10**
- 6.1 Similarity Search Theory . . . 10
- 6.2 Why FAISS? . . . 10
- 6.3 Implementation . . . 10
- 6.4 Summary . . . 11

7 User Interface **12**
- 7.1 Design Principles . . . 12
- 7.2 Query Workflow . . . 12
- 7.3 Filtering and Sorting . . . 12
- 7.4 Result Display . . . 12
- 7.5 Privacy and Offline Execution . . . 13

8 Performance and Evaluation Challenges **14**
8.1 Evaluation Without Labels . . . 14
8.2 Qualitative Evaluation . . . 14
8.3 Category Inference Evaluation . . . 14
8.4 Responsiveness and Latency . . . 15
8.5 Scalability Considerations . . . 15
8.6 Limitations of Offline Evaluation . . . 15
8.7 Summary . . . 15

9 Conclusion **16**
9.1 Discussion . . . 16
9.2 Conclusion . . . 16
9.3 Reflection . . . 17

References **18**

A Glossary of Mathematical Symbols **19**

B Glossary of ML and AI Terms **20**

C Figures **21**

D Sentence Embeddings **29**

E Zero-Shot Classification **31**

F Similarity Search with FAISS **33**

G Confidence Filtering **35**

H Fallback Classification with Keywords **37**

1 Introduction

1.1. Motivation

This project explores the feasibility of deploying local machine learning models for intelligent information retrieval in local environments. As a case study, a fully local book recommendation system is developed. It leverages Natural Language Processing (NLP) techniques to analyze both book descriptions and natural language queries from users, enabling semantically meaningful, content-based recommendations.

In contrast to cloud-based systems that depend on centralized APIs and remote computation, this solution demonstrates a standalone, offline setup. All data processing — including text embedding, category inference, indexing, and query resolution — occurs on the user's device, ensuring that no personal data leaves the local environment.

The core goal is to evaluate whether lightweight transformer-based models, specifically sentence transformers like MiniLM, can provide reliable and personalized recommendations within such constraints. The system incorporates semantic embedding, category inference through zero-shot classification, vector similarity search via FAISS, and a user-friendly Streamlit interface. These components collectively address the broader question of how accessible and effective locally hosted AI tools 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 relevant books?*
3. *How can genre categories be inferred and used to improve indexing and filtering?*
4. *What are the practical limitations of a local, content-only recommendation system?*

These sub-questions form the conceptual framework for the analysis presented throughout the synopsis.

1.3. Link to code repository

The code is public on GitHub: https://github.com/Cal-ly/LLM-Book-Recommender.

2

Methodology and Structure

This project uses a research-driven prototype methodology to evaluate the feasibility of deploying semantic book recommendation systems powered entirely by locally running machine learning models. Rather than building a commercial product, the objective was to explore performance, usability, and effectiveness under offline constraints.

2.1. Research Approach

The methodology combines literature review, modular implementation, and empirical testing. Each decision supports the research question and sub-questions (see Section 1.2).

- **Literature Review:** Focused on content-based recommender systems, sentence embeddings, zero-shot classification, and design principles Geron, 2022.
- **Implementation Tools:**
  - `Python` with `pandas` for data pipelines.
  - `sentence-transformers` (MiniLM-L6-v2) for generating semantic embeddings.
  - `facebook/bart-large-mnli` for zero-shot classification of genres.
  - `FAISS` for efficient vector indexing and retrieval.
  - `Streamlit` for a local-first UI interface.
- **Data Preparation:** Metadata was enriched and filtered using hybrid classification with confidence thresholds and fallback logic.
- **Empirical Testing:** Semantic quality and responsiveness were evaluated via real-world queries and edge-case prompts.

2.2. Evaluation Criteria

No labeled test set was available, so evaluation focused on:

- **Semantic relevance** — how well results match query intent.
- **Responsiveness** — ability to respond quickly on CPU-only hardware.
- **Filtering utility** — impact of genre/rating filters.
- **Offline execution** — system runs without network access.

These metrics align with sub-questions 1, 2, and 3.

2.3. Structure of the Synopsis

2.3.1. Main Chapters

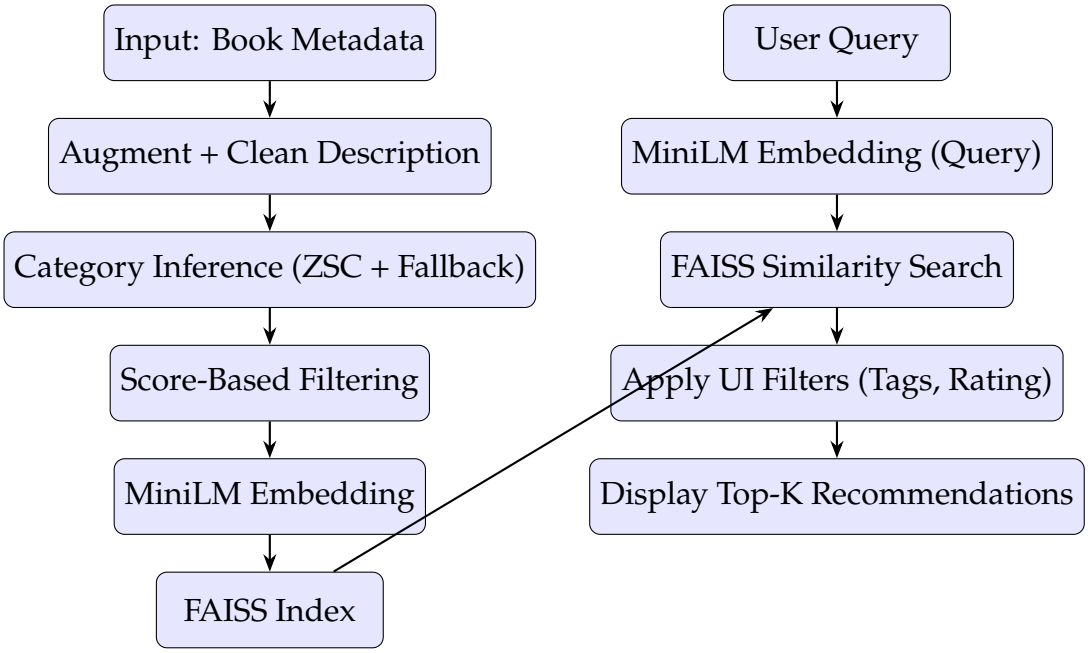
Each core system component is described in its own chapter, addressing one or more of the research sub-questions.

2.3.2. Appendices

Supporting materials include example code, glossary terms, and diagrams that expand on key technical concepts.

2.4. Application Architecture

The overall system is modular and fully local:



3

Dataset Exploration

The project began with the public dataset by Castillo (2025), containing metadata for over 6,800 books. Initial inspection revealed missing or inconsistent fields such as authorship, categories, and descriptions. A modular preprocessing pipeline was built to clean, augment, and prepare the data for semantic modeling.

3.1. Dataset Overview

The original dataset (`books.csv`) included fields such as:

- `isbn13`, `title`, `subtitle`, `authors`
- `categories`, `description`, `published_year`
- `average_rating`, `num_pages`

Key fields were combined and engineered into a unified `full_title`, and several boolean `has_*` flags were created for inspection and filtering.

3.2. Data Cleaning and Augmentation

Cleaning and augmentation were performed in multiple stages:

1. **Initial Checks:** Detected and logged missing or invalid entries.
2. **OpenLibrary Augmentation:** Filled in missing values such as `authors`, `num_pages`, and `thumbnail`. Introduced `subjects`.
3. **Google Books API:** Prioritized short or missing descriptions and added alternate fields (e.g., `description_google`).
4. **Field Comparison:** Logged mismatches in fields like title and author between sources. Created `alt_*` fields where minor but significant discrepancies were found.
5. **Final Merging:** Consolidated `categories`, `subjects`, and `categories_google` into a cleaned `final_categories` field.

Records with less than 9 words in the final description were removed, reducing the dataset from 6,810 to 6,572 entries.

3.3. Feature Engineering

New fields were engineered to support classification and filtering:

- `words_in_description` — token count of description
- `description_length` — character count of enriched description

- `has_*` flags — completeness indicators for filtering

These were used both for data readiness assessments and visualization.

3.4. Exploratory Analysis

Visual and statistical analysis was used to inform thresholds and highlight data issues:

- **Missing Values:** `openlib_values_heatmap.png`
- **Rating Distribution:** `rating_distribution.png`
- **Publication Year:** `publication_year_distribution.png`
- **Category Frequency:** `top_categories.png`
- **Short Descriptions:** `less_than_50_words_description.png`
- **Metadata Conflicts:** `reexp_mismatch_counts.png`

These figures helped guide filtering strategies and offered insight into metadata quality.

3.5. Outcome

After augmentation and filtering, 6,572 books remained. These were passed to the category inference pipeline (Chapter 4), where further refinement reduced the set to 5,160 high-confidence entries suitable for semantic embedding and indexing.

4

Category Inference

To support genre-aware filtering and embedding, each book in the dataset is automatically labeled with one or more thematic categories. This process combines transformer-based zero-shot classification with a curated keyword-based fallback strategy, confidence-based filtering, and semantic cleaning. The refined categories are ultimately used for filtering, indexing, and interface interaction.

4.1. Zero-Shot Classification with BART-MNLI

Each book is first enriched with an `augmented_description` field, combining its full title, author, publication year, and description. These are then fed into a zero-shot classification model, `facebook/bart-large-mnli` Facebook AI, 2025, using Hugging Face's `pipeline` for multi-label inference across 13 candidate categories (e.g. *Fantasy*, *Love*, *Non-fiction*).

Predictions below a confidence threshold of 0.4 are discarded. Each book retains only the labels with strong relevance, based on model probabilities.

4.2. Fallback Keyword Enrichment

To improve recall and thematic depth, fallback keywords are used when the model fails to detect relevant categories. Each category has a corresponding keyword list (e.g. "`magic`", "`orc`", "`griffin`" for *Fantasy*; "`vampire`", "`haunted`", "`exorcism`" for *Horror*).

Fallback labels are only added if the model's predictions do not already include them. In total, fallback logic affected 4665 out of 6572 rows.

4.3. Refinement, Merging, and Metrics

To reduce redundancy and overlap, closely related categories are merged:

- **Biography + History** → *Historical*
- **Philosophy + Poetry** → *Philosophy & Poetry*
- **Suspense + Detective** → *Mystery*

Per-category precision, recall, and F1 scores are calculated using known metadata labels for validation. Visualizations and metric CSVs are generated for review.

4.4. Confidence Analysis and Filtering

Each row is evaluated using the following confidence metrics:

- `max_score`: Highest label confidence
- `filtered_avg_score`: Average score excluding labels under 0.2
- `score_std`: Standard deviation across all label scores
- `num_categories`: Total confident categories retained

To prepare for indexing, only rows satisfying all of the following are retained:

- Description length ≥ 200 characters
- `filtered_avg_score` ≥ 0.2
- `max_score` ≥ 0.4
- At least one confident category

This reduced the dataset from 6572 to 5160 books.

4.5. Length vs Confidence Correlation

To verify the impact of text richness, correlation between `description_length` and `average_score` was analyzed. The results:

- Pearson correlation: $r = 0.398$ ($p < 0.0001$)
- Spearman rank: $\rho = 0.261$ ($p < 0.0001$)

This indicates a moderate positive relationship between description length and model confidence, validating the filtering step as both qualitative and statistically grounded.

5

Text Embedding

The semantic recommender relies on converting each book entry into a dense vector representation, allowing high-dimensional similarity comparison. This is achieved through sentence embeddings using a pretrained transformer model. By embedding descriptive text into vector space, the system can retrieve semantically related books using efficient mathematical distance metrics.

5.1. Sentence Embedding Theory

Sentence embeddings are fixed-size vector representations of variable-length text sequences. They capture semantic similarity, enabling meaningful comparisons via cosine similarity or L2 distance. Formally, for an input string t, the embedding function f maps it into $\mathbb{R}^d$:

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

The model `all-MiniLM-L6-v2` from Hugging Face, 2025 is used.
It is a lightweight transformer from the `sentence-transformers` library. It provides a strong trade-off between performance and inference speed, producing 384-dimensional vectors suitable for local, offline use.

5.2. Embedding Implementation

Embeddings were generated from the refined and filtered dataset `books_indexed.csv`, which includes a special `search_text` field constructed for each row:

```
Title: {full_title}. Author: {authors}. Description: {description}.
```

This composite text ensures that not only the description but also the title and author are considered in the embedding. The embedding pipeline performs the following:

1. Loads `search_text` entries from `books_indexed.csv`.
2. Encodes each entry using `SentenceTransformer`.
3. Outputs a NumPy array of shape $(n, 384)$, where $n = 5,160$.
4. Adds the matrix to a `FAISS` index for fast similarity lookup.
5. Saves associated metadata to `books_indexed_with_embeddings.csv`.

5.3. Embedding Configuration

The embedding system is configured as follows:

- **Model:** `all-MiniLM-L6-v2`
- **Dimension:** 384
- **Library:** `sentence-transformers`
- **Distance Metric:** L2 distance (used by FAISS)
- **Backend:** CPU inference
- **Input Field:** `search_text`
- **Entries Embedded:** 5,160 books

5.4. Summary

Embedding is the core enabler of semantic recommendation. Using a compact transformer model to generate high-quality vector representations from full book metadata, the system maintains relevance without relying on user ratings, tags, or prior behavior. This allows for query matches using purely content-based inference.

6

Vector Similarity Search with FAISS

Once book and query texts are embedded into dense vector space, the task becomes identifying which book vectors are most similar to a given query vector. This is accomplished using FAISS — Facebook AI Similarity Search AI, 2025, a library optimized for fast vector comparisons.

6.1. Similarity Search Theory

Given a user query q and a set of embedded book vectors $\{\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_n\}$, the system computes similarity using 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 entry
- The top k results with the smallest distances are returned

This assumes semantically similar inputs are located close together in vector space — an assumption supported by transformer-based embeddings.

6.2. Why FAISS?

FAISS is designed for efficient similarity search across high-dimensional data. It is widely used in recommendation, retrieval, and clustering tasks.

Key advantages include:

- **Speed:** Real-time query responses even with thousands of entries.
- **Simplicity:** Flat indexes (e.g. `IndexFlatL2`) support exact search.
- **Offline Support:** FAISS runs locally on CPU, making it ideal for private, embedded deployments.

In this project, we use `IndexFlatL2` for exact nearest-neighbor search using Euclidean distance.

6.3. Implementation

After embedding all book entries using `MiniLM-L6-v2`, the following FAISS-based steps are applied:

1. A FAISS index is created with `faiss.IndexFlatL2(384)`.
2. All 5,160 book vectors are added to the index.
3. The index is saved as `embeddings/index.faiss`.
4. At runtime:
    - A user query is embedded using the same transformer model.
    - The top k closest vectors are retrieved via FAISS.
    - Metadata is joined from `books_indexed_with_embeddings.csv`.

This provides fast and accurate lookup for semantically similar books without relying on collaborative signals.

6.4. Summary

FAISS enables efficient real-time recommendations by using a dense vector index. Together with MiniLM-based embeddings, it allows natural language queries to find relevant books using only their content.

7

User Interface

Although not a machine learning component, the user interface (UI) enables users to interact with the system naturally and intuitively. It is implemented in `Streamlit`, a lightweight Python framework suitable for local-first applications.

7.1. Design Principles

The UI emphasizes simplicity and all inference, vector search, and filtering are executed locally.

Design goals:

- Enable concept-based, natural language search.
- Support filtering and sorting using metadata (rating, tags).
- Eliminate external dependencies, aside from optional cover lookups.

7.2. Query Workflow

Users submit free-form queries (e.g., `"Books about survival in a post-apocalyptic world"`) via a single input box. The process:

1. The query is embedded with the same MiniLM model.
2. The vector is passed to FAISS to find the 60 nearest neighbors.
3. Results are filtered and sorted.
4. 6 book cards are shown per page.

7.3. Filtering and Sorting

Users can interactively refine results using:

- **Genre/Tag Filter:** Multi-select from inferred categories.
- **Rating Sort:** Toggle ascending/descending by average rating.
- **Pagination:** Navigate in groups of 6 results.

Filters are applied after semantic matching.

7.4. Result Display

Each book is shown as a card with:

- **Cover Image:** Loaded from URL, with local fallback.
- **Title and Author**

- **Average Rating** as stars
- **Short Description** (first 300 characters)
- **Refined Categories**

7.5. Privacy and Offline Execution

The system offers full offline functionality:

- No data is collected or transmitted.
- No login, cookies, or tracking.
- Fully functional without internet access.

8

Performance and Evaluation Challenges

This chapter discusses how the system's performance was evaluated in the absence of user interaction or labeled ground-truth data. Traditional metrics such as accuracy or recall are not applicable to semantic similarity or unsupervised recommendation. As such, evaluation was divided into subjective, statistical, and system performance components.

8.1. Evaluation Without Labels

Unlike supervised learning, this recommender system does not predict a fixed class label or known output. It embeds books and queries into a shared semantic space and retrieves nearest neighbors based on vector distance. As a result:

- There is no single "correct" recommendation for a given query.
- Metrics like precision and recall do not apply in the absence of labeled matches.
- Offline systems cannot collect feedback to infer satisfaction or click-through rate.

8.2. Qualitative Evaluation

Evaluation relied on practical usage testing and subjective judgment:

- A wide variety of test queries were manually issued.
- The semantic alignment of results was judged by human inspection.
- Edge cases such as extremely short queries or unusual concepts (e.g., "existential loneliness in space") were tested.

These informal tests confirmed that recommendations typically aligned well with the user's intent, especially when descriptions were rich and the classification succeeded.

8.3. Category Inference Evaluation

Although no labeled test set existed for genre prediction, basic model evaluation was conducted:

- Confidence scores were retained for each category prediction.
- Categories were filtered based on a score threshold (≥ 0.4).
- F1-score, precision, and recall were plotted per category using a trusted subset.
- Keyword-based fallback performance was analyzed independently.

This analysis confirmed that certain categories (e.g. *Fantasy*, *Love*) were well-captured by the model, while others (e.g. *Philosophy & Poetry*) relied more heavily on keyword

support.

8.4. Responsiveness and Latency

Runtime performance was measured on a CPU-based consumer laptop with no GPU acceleration:

- **Embedding a single query:** < 0.2 seconds (MiniLM-L6-v2)
- **Top-60 vector search:** < 10 milliseconds (FAISS `IndexFlatL2`)
- **UI rendering (6 results/page):** < 2 seconds including image fallback

The system was fully usable in real time, validating its suitability for local deployment.

8.5. Scalability Considerations

Scalability remains a practical concern, especially if the dataset grows:

- **Indexing:** FAISS is fast for exact search with thousands of vectors, but approximate methods like HNSW or IVF may be needed at scale.
- **Embedding refresh:** All books must be re-encoded when changing the model or dataset.
- **Post-filtering:** Filtering and sorting happen after retrieval and may become slow for large result sets.

For a production system, batching, vector quantization, or UI caching would be necessary.

8.6. Limitations of Offline Evaluation

The offline, user-free nature of the prototype limits what can be empirically measured:

- **Relevance:** Only developer judgment is available.
- **Discovery:** No tracking exists to measure novelty or diversity.
- **Bias Detection:** Over-representation of popular authors or themes may not be obvious without user feedback.

These limitations are expected in an exploratory prototype and suggest directions for future iterations.

8.7. Summary

Performance evaluation focused on semantic quality, speed, and usability. While classical metrics were not applicable, confidence filtering, qualitative testing, and runtime benchmarks collectively demonstrate that the system is performant, accurate, and well-aligned with its offline-first goals.

9 Conclusion

9.1. Discussion

This project demonstrates that modern transformer-based NLP models can power effective book recommendation purely from textual content. Without user profiles or collaborative filtering, the system performs semantic matching based on enriched descriptions and lightweight local inference.

Through category inference, vector embedding, and high-speed similarity search, the system enables offline, local recommendations — a viable alternative to cloud-based recommenders.

9.2. Conclusion

The following sub-questions were addressed:

1. What techniques exist for embedding text into meaningful vectors?
Pretrained transformer models like `MiniLM-L6-v2` from the `sentence-transformers` library produce dense, context-aware sentence embeddings. These vectors represent semantic content and enable meaningful similarity comparisons.

2. How can vector similarity be used for finding similar books?
Books and queries are mapped into the same vector space. FAISS is used to compute nearest neighbors based on L2 distance, retrieving top-matching books efficiently from a corpus of 5,160 entries.

3. How can genre categories be inferred and used to improve indexing and filtering?
Zero-shot classification with `facebook/bart-large-mnli` was used to assign semantic tags. Fallback keywords and confidence filtering ensured each book had at least one interpretable label, improving both search relevance and UI usability.

4. What are the limitations of a local, content-only recommender?
The system lacks personalization, user learning, and click feedback. Results are driven solely by content similarity. However, these trade-offs are offset by strong privacy, transparency, and independence from third-party APIs.

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 work has shown that sentence embeddings, zero-shot labeling, and vector search can be combined into a compact, offline-first recommender system. Users can issue free-form queries and explore books by theme, genre, or rating — all without internet connectivity or external dependencies.

9.3. Reflection

The project provided insight into applied NLP, recommender architecture, and the challenges of building local-first AI applications. Key takeaways:

- **Quality of input affects all stages:** Rich, well-structured descriptions significantly improve category inference and search results.
- **Fallback strategies increase coverage:** When zero-shot confidence is low, keyword-based tagging preserves semantic grounding.
- **Local-first ML is practical:** FAISS, MiniLM, and Streamlit form a strong open-source stack for real-time semantic querying.
- **Trade-offs remain:** While private and responsive, the lack of personalization limits advanced use cases.

With more time, the project could be extended to:

- Compare multiple embedding models (e.g., MPNet, SBERT).
- Implement metadata-based re-ranking for top-k results.
- Incorporate user feedback or interactive scoring for future personalization.

Ultimately, the system proves that locally hosted NLP models can provide meaningful recommendations using only text and a consumer-grade CPU.

References

AI, M. (2025). *Faiss facebook ai similarity search*. Retrieved May 8, 2025, from https://faiss.ai/

Facebook AI. (2025). *Facebook/bart-large-mnli*. Retrieved May 8, 2025, from https://huggingface.co/facebook/bart-large-mnli

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

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

A

Glossary of Mathematical Symbols

This appendix provides a reference for the mathematical notation used throughout the synopsis.

Symbol	Meaning
$\vec{v}$	Vector (e.g., embedding of a sentence or query)
$\mathbb{R}^d$	d-dimensional real-valued vector space
$\|\vec{a} - \vec{b}\|_2$	Euclidean (L2) distance between vectors $\vec{a}$ and $\vec{b}$
$f(t)$	Embedding function applied to text t
t	Input text (sentence or description)
q	Query vector
b_i	ith book vector
k	Number of nearest neighbors retrieved from the index
ρ	Spearman rank correlation coefficient
r	Pearson linear correlation coefficient

B

Glossary of ML and AI Terms

This appendix provides definitions of key machine learning and AI terms used throughout the synopsis.

Term	Definition
Embedding	A numerical vector representing the semantic meaning of a text input
Sentence Transformer	A transformer model that converts full sentences into vector embeddings
FAISS	Facebook AI Similarity Search: a library for fast vector search
L2 Distance	Euclidean distance metric between two points in vector space
Zero-Shot Classification	Labeling data without task-specific training, using general-purpose models
Fallback Keywords	Curated terms used to infer categories when model confidence is insufficient
MiniLM	A small, efficient transformer model for sentence embedding tasks
BART-MNLI	A pretrained model for natural language inference used in category prediction
Refined Categories	Cleaned and merged genre labels derived from model output or fallback logic
Streamlit	A Python framework for creating web-based data applications
Filtered Average Score	Average confidence of category predictions above a threshold (e.g., 0.2)
IndexFlatL2	A FAISS index type that performs exact nearest-neighbor search using L2 distance

C

Figures

This appendix presents the figures generated throughout the data processing pipeline. Each figure is introduced in the order it was produced, along with a short explanation of what it represents and how it informed decisions in the pipeline.

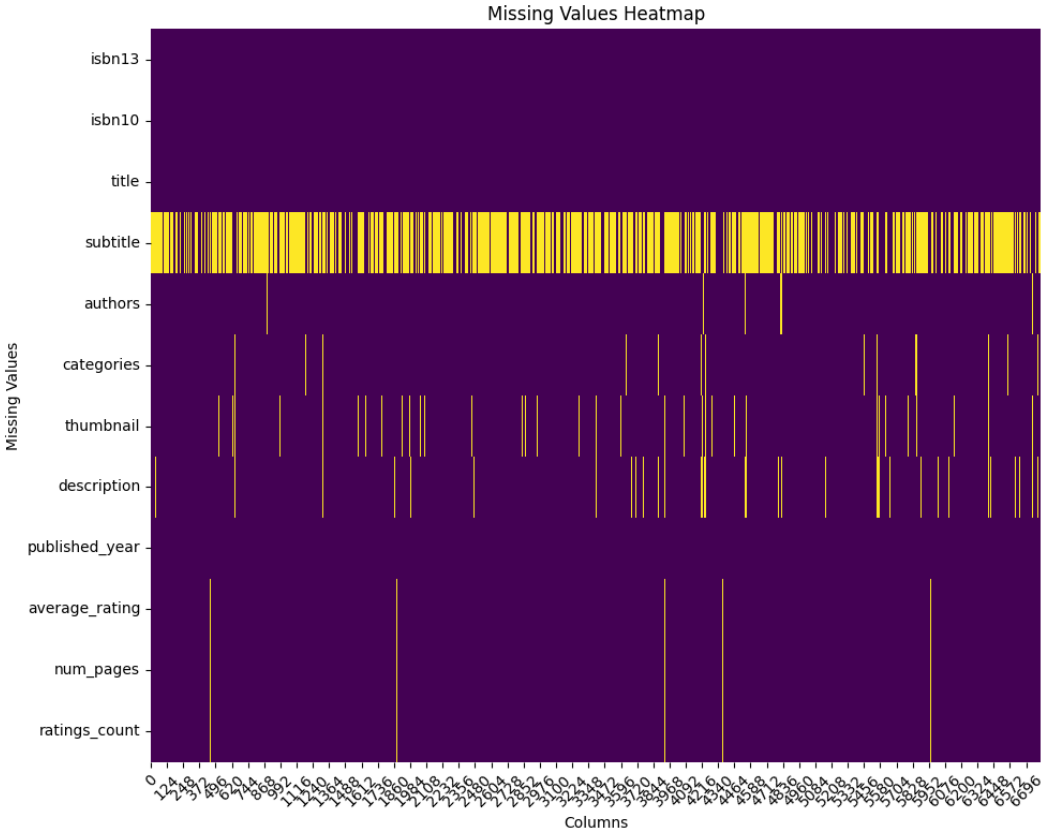


Figure C.1: Missing Values Heatmap: Visualizes the density of missing fields in the original dataset. The heatmap highlights common gaps such as missing subtitles, authors, and thumbnails, prompting the need for augmentation.

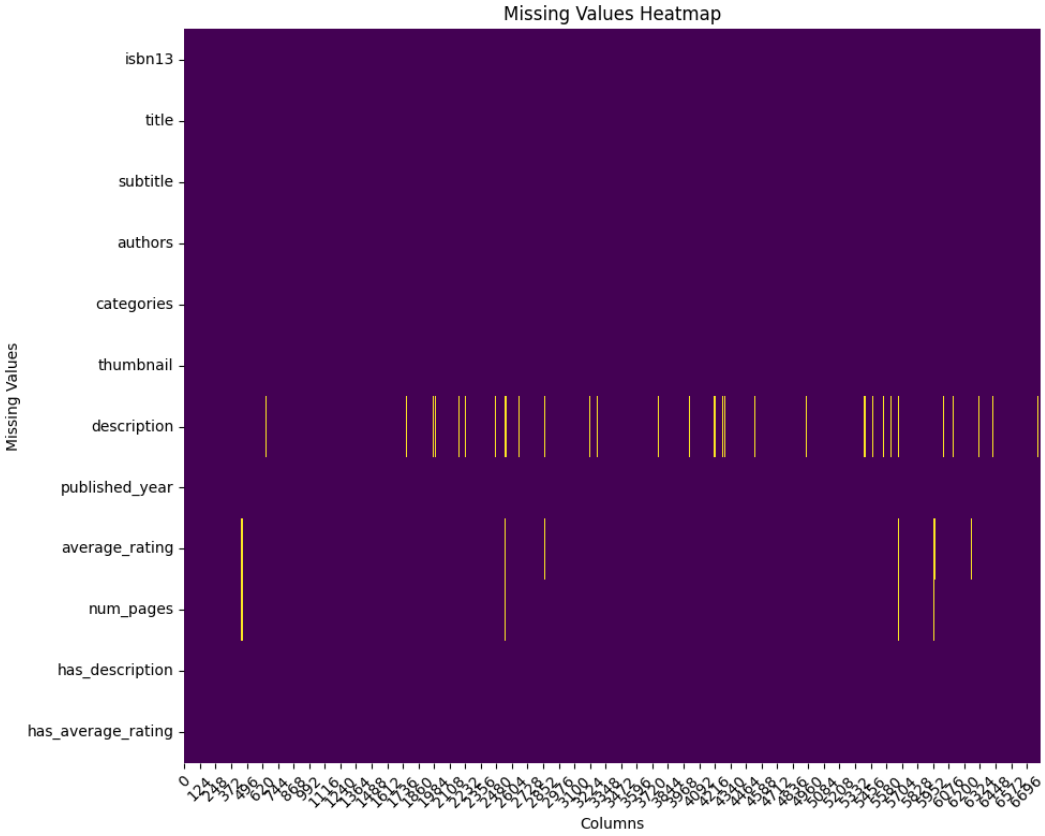


Figure C.2: OpenLibrary Completion Heatmap: Displays the remaining missing values after OpenLibrary augmentation. It shows improvements in coverage of fields like subtitles, page counts, and published years.

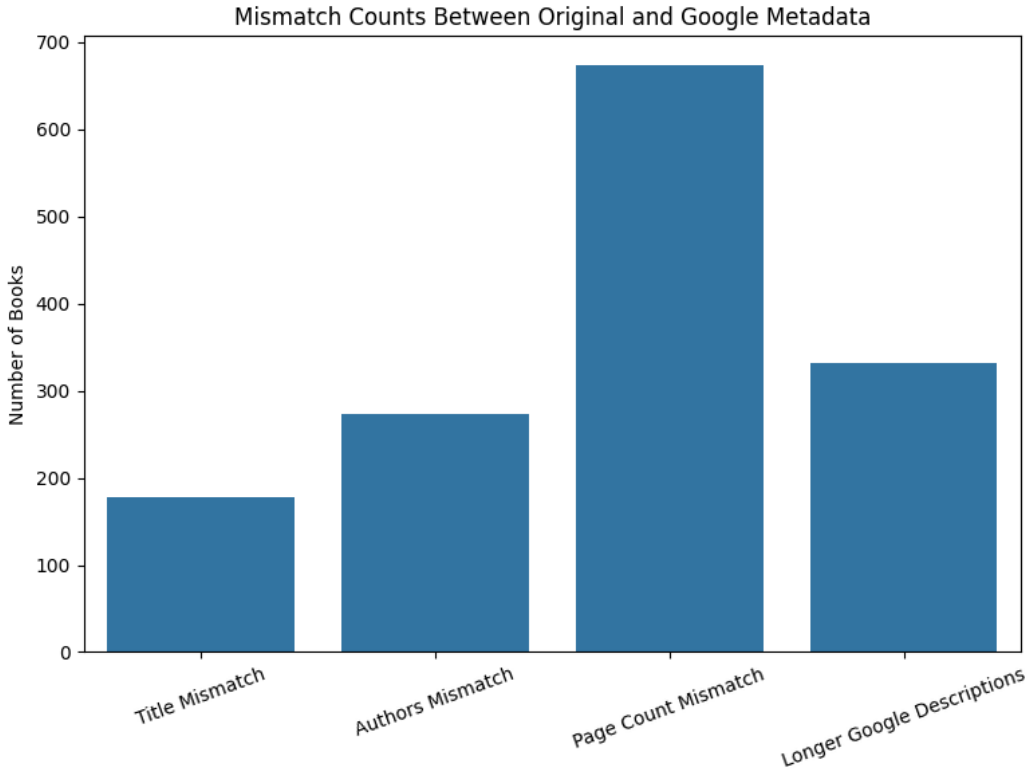


Figure C.3: Metadata Conflict Counts: A bar chart showing the number of books with mismatches in title, author, or page count across sources. This justified creation of alternative metadata fields.

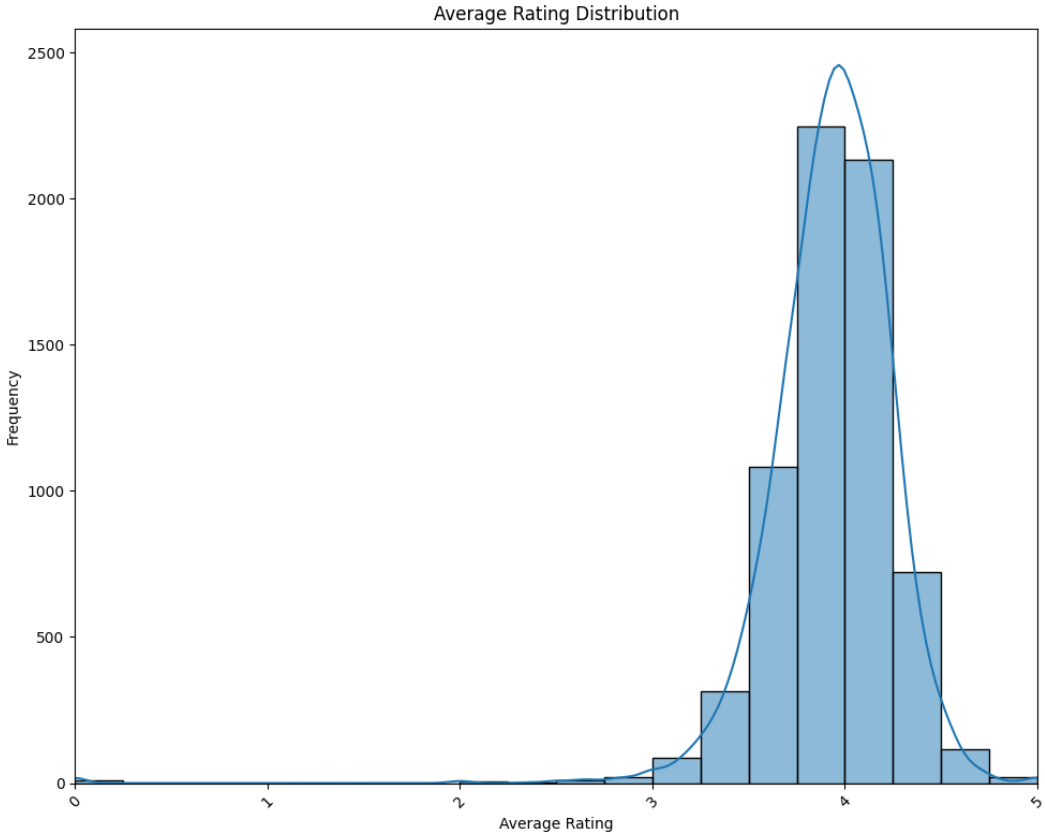


Figure C.4: Rating Distribution: Histogram of average book ratings. The skew toward higher ratings is typical for user-generated content and informed filter default settings.

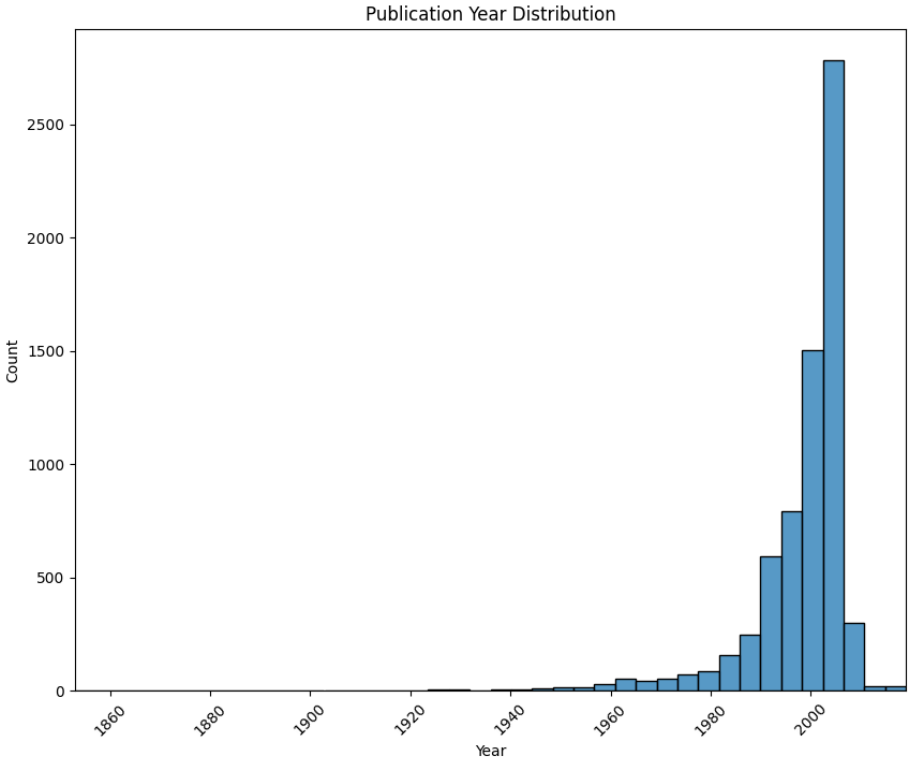


Figure C.5: Publication Year Distribution: Shows the number of books published per year. The majority are post-2000, indicating a recency bias that may impact topic coverage.

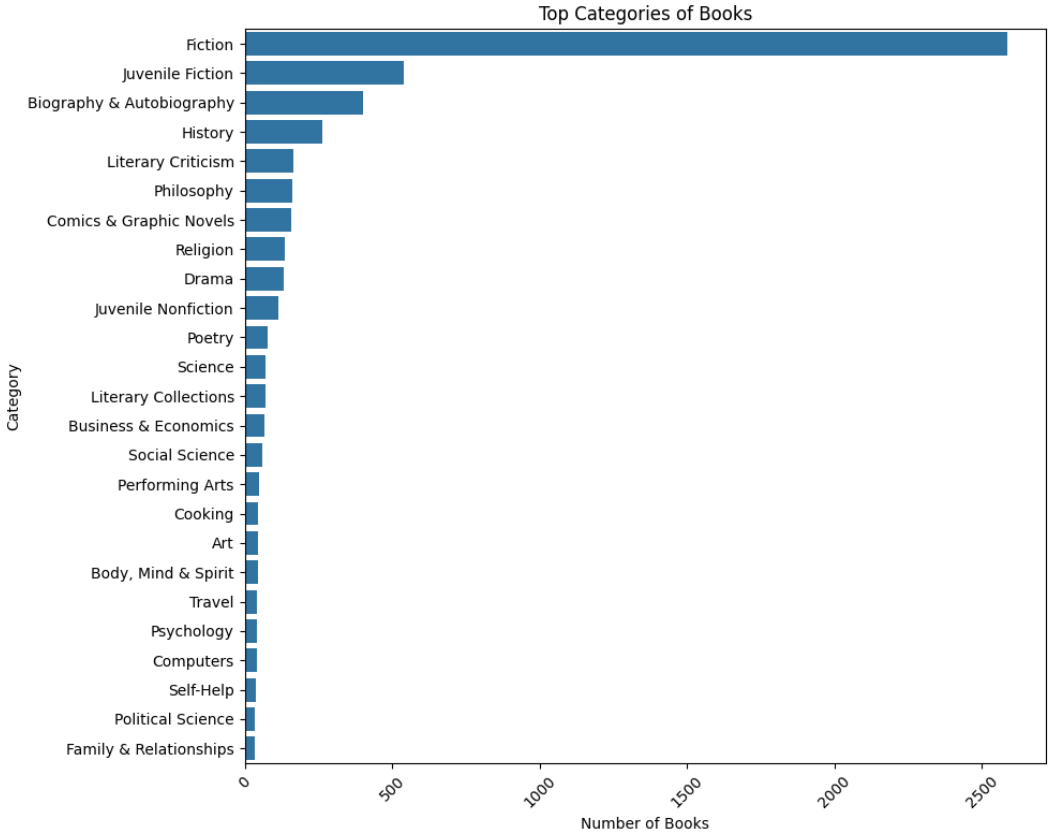


Figure C.6: Top Categories: A bar chart of the most frequent category labels in the original dataset. This informed the definition of candidate labels for zero-shot classification.

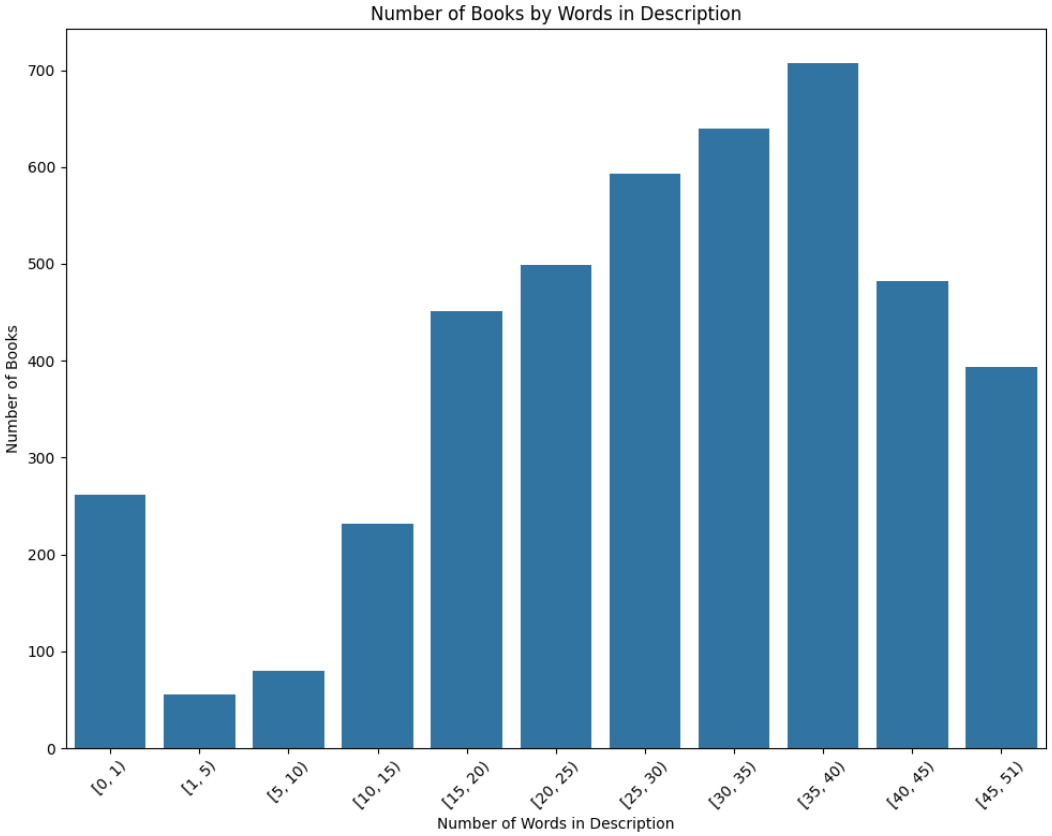


Figure C.7: Books with Short Descriptions: Distribution of books with fewer than 50 words in their descriptions. These entries were flagged for enrichment or removal.

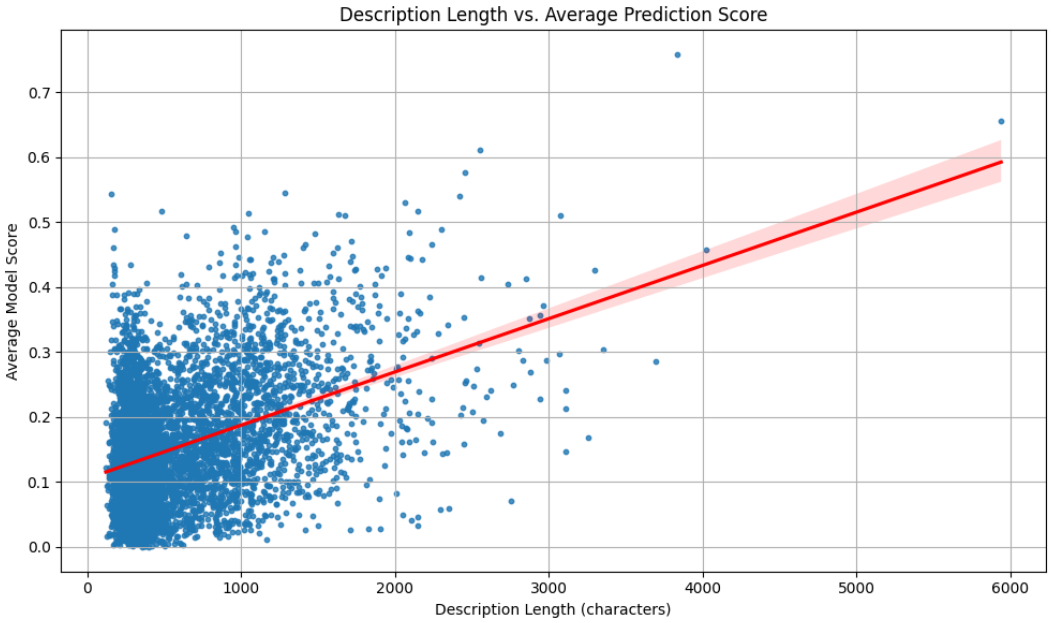


Figure C.8: Description Length vs. Confidence Score: A regression plot showing the positive correlation between description length and average model confidence. This supported the threshold of 200 characters.

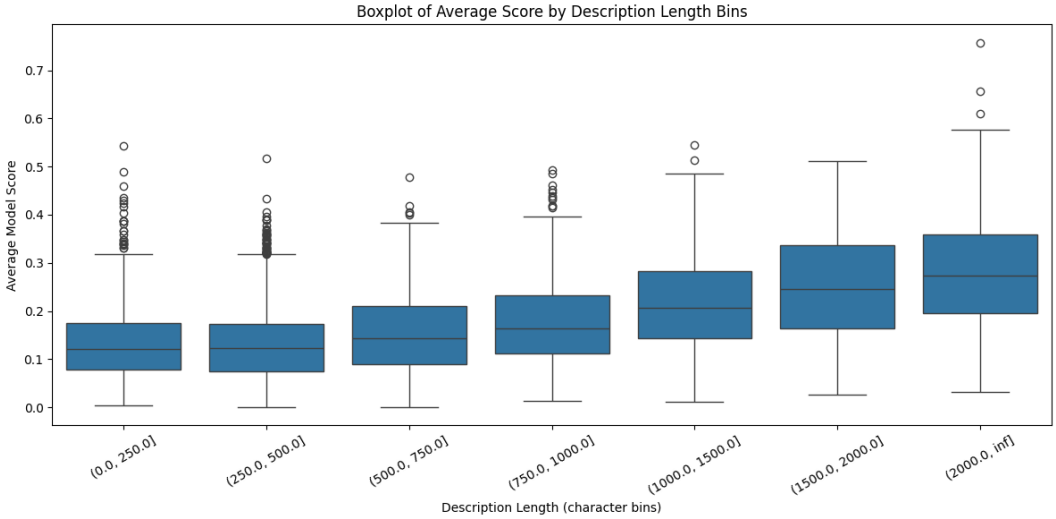


Figure C.9: Score by Length Bin: A boxplot displaying score variation across binned description lengths. Helps visualize consistency and reliability of model outputs by text richness.

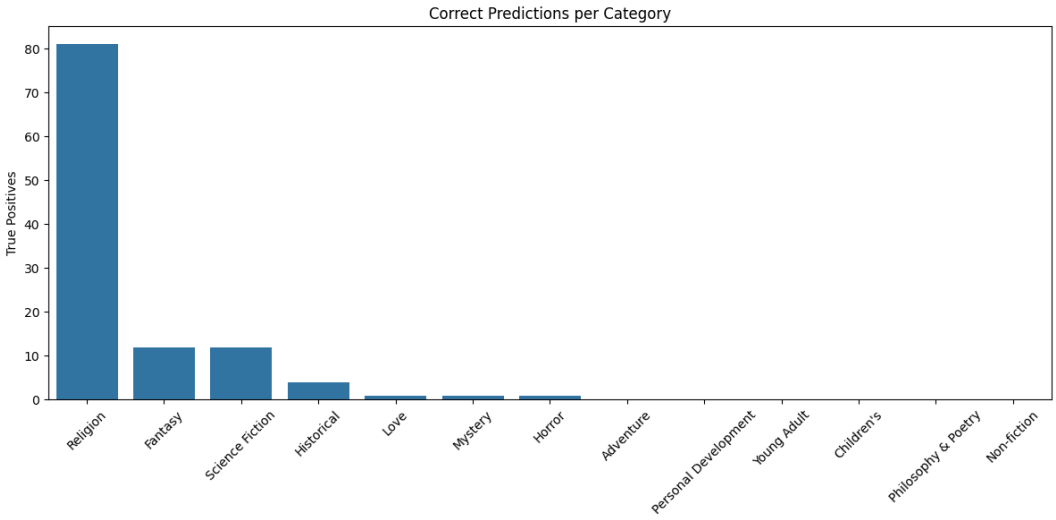


Figure C.10: Prediction Matches per Category: Bar chart of true positives per category. Reflects model strength in domains like Fantasy and Love and highlights where fallback logic is critical.

D

Sentence Embeddings

Sentence embeddings are dense vector representations of entire sentences, designed to capture semantic meaning. Unlike traditional bag-of-words models, embeddings preserve word order and contextual meaning, allowing for nuanced comparisons between texts.

Intuition

Where classic keyword-based systems rely on surface matches, sentence embeddings map similar meanings to nearby points in vector space. For instance:

`"A tale of survival in space"`

`"Story about astronauts facing danger beyond Earth"`

Though these phrases share few keywords, their embeddings are close in high-dimensional space.

Transformer-Based Embeddings

Transformer models such as BERT, RoBERTa, and MiniLM process text using self-attention mechanisms. They generate contextual embeddings — meaning the word `bank` in "river bank" and "investment bank" will have different vector representations.

Sentence transformers apply pooling strategies (e.g., mean pooling) over token embeddings to produce a single vector per sentence.

Dimensionality and Use

In this project, the model `MiniLM-L6-v2` produces 384-dimensional embeddings:

$$\vec{v} \in \mathbb{R}^{384}$$

These embeddings are used to:

- Represent books (title + description + author)
- Represent user queries
- Enable similarity search via FAISS

Similarity Metrics

Cosine similarity and L2 (Euclidean) distance are common metrics. In this system, FAISS uses L2 distance:

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

Smaller distances imply stronger semantic similarity.

Benefits and Limitations

- **Pros:** Captures meaning beyond exact words; enables cross-domain generalization; fast inference with MiniLM.
- **Cons:** Embedding quality depends on input quality; short or vague inputs yield weak vectors.

E

Zero-Shot Classification

Zero-shot classification is a machine learning technique that allows a model to assign labels to input text without having seen labeled examples for those specific classes. It relies on models trained on general-purpose tasks such as Natural Language Inference (NLI).

Why Zero-Shot?

In this project, we did not have human-labeled genres for the books. Instead, we defined a set of target categories (e.g., *Fantasy*, *Science Fiction*, *Historical*) and used a pretrained NLI model to decide whether a description "entails" each candidate label.

NLI-Based Classification

The model used was `facebook/bart-large-mnli`, a transformer trained on premise-hypothesis relationships.

Given:

- Premise: the book description
- Hypothesis: "This book is *[label]*"

The model evaluates whether the hypothesis is entailed by the premise. A high entailment probability means the label is likely correct.

Multi-Label Inference

Zero-shot classification supports multi-label predictions. A single description may be associated with multiple categories. Only predictions above a threshold (e.g., 0.4) were retained.

Advantages

- No training required on project-specific data
- Easily extendable to new label sets
- Strong generalization using a well-trained NLI base

Limitations

- Performance depends on hypothesis phrasing (e.g., "This book is *fantasy*" vs "This story involves *fantasy*")
- May confuse overlapping or abstract categories

- Computationally slower than simple keyword matching

Zero-shot classification enabled this project to assign interpretable categories to books without manual labeling, supporting both UI filtering and statistical analysis.

F

Similarity Search with FAISS

Similarity search is the task of finding items most similar to a given query in a high-dimensional vector space. This project uses FAISS (Facebook AI Similarity Search) to perform fast, exact nearest-neighbor lookups over book embeddings.

Motivation

Books and queries are embedded into the same semantic vector space. To recommend relevant books, the system retrieves the closest vectors (books) to a given query vector.

What is FAISS?

FAISS is a C++/Python library developed by Meta for fast search across large collections of dense vectors. It supports both exact and approximate search methods.

Indexing Method Used

This system uses the exact search index:

- `IndexFlatL2` — Computes L2 (Euclidean) distance between the query and all book vectors
- Suitable for small to medium datasets (~5,000 vectors)
- No compression or quantization

Search Workflow

1. Book metadata is converted to `search_text` and embedded
2. Vectors are added to a FAISS index
3. At query time, the user input is embedded
4. FAISS returns top-k nearest neighbors
5. Metadata is retrieved from the indexed CSV

Distance Metric

FAISS computes squared L2 distance:

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

Lower values indicate higher semantic similarity.

Benefits

- Extremely fast for exact matching on CPU
- Integrates easily with NumPy and PyTorch
- Fully offline and open source

Alternatives

For larger datasets, approximate methods like HNSW, IVF, or PQ (quantization) may be more scalable but require tuning.

FAISS was ideal for this project due to its simplicity and performance at the given scale.

G

Confidence Filtering

Confidence filtering is the process of retaining only high-certainty predictions from a model. In this project, it is used to ensure that category labels assigned to books are reliable.

Motivation

Zero-shot classification produces a confidence score for each category. However, not all scores are meaningful — low-confidence predictions can lead to noisy or irrelevant labels.

Metrics Used

Three metrics were computed for each book's classification output:

- **max_score** — the highest confidence score among all categories
- **filtered_avg_score** — average confidence score for predictions above a threshold (e.g., 0.2)
- **score_std** — standard deviation of all confidence scores

Filtering Strategy

To retain high-quality entries for indexing and UI use, books were required to meet all of the following:

- `description_length` ≥ 200 characters
- `filtered_avg_score` ≥ 0.2
- `max_score` ≥ 0.4
- At least one predicted category

Why Multiple Metrics?

- **max_score** ensures at least one strong category signal
- **filtered_avg_score** avoids noisy averages dominated by low scores
- **score_std** (analyzed but not thresholded) helps detect ambiguous predictions

Impact

These thresholds reduced the dataset from 6,572 to 5,160 entries, each with well-supported category labels and semantically rich descriptions. This filtering greatly improved the quality of recommendations.

Confidence filtering balances recall and precision, ensuring that the final indexed dataset is useful and trustworthy.

H

Fallback Classification with Keywords

While zero-shot classification captured many semantic categories with high accuracy, some descriptions were too vague or short for the model to assign reliable labels. To compensate, a fallback strategy based on keyword matching was added.

Motivation

Some books lack rich descriptions or use metaphoric language. For example, a horror book might avoid directly mentioning "ghosts" or "monsters" but still belong to the genre. Keywords help catch these implicit themes.

Keyword Lists

Each predefined category was associated with a curated list of terms. For instance:

- **Fantasy:** `magic, wizard, dragon, elf, spell`
- **Science Fiction:** `space, alien, robot, galaxy, dystopia`
- **Love:** `romance, passion, heartbreak, relationship`

These lists were handcrafted and refined iteratively during testing.

Matching Logic

After zero-shot prediction:

- Each description (or augmented description) was lowercased.
- Keyword presence was checked using regular expressions.
- Fallback labels were added only if not already predicted by the model.

Benefits

- Increases coverage of categories
- Captures under-expressed themes in sparse descriptions
- Keeps control in hands of the developer (interpretable logic)

Drawbacks

- Sensitive to phrasing and vocabulary
- May overfit to genre stereotypes (e.g., “dragon” always implies Fantasy)
- Requires manual tuning and maintenance

Fallback classification ensures that every book receives at least one thematic label, improving both UI filter functionality and downstream semantic search quality.