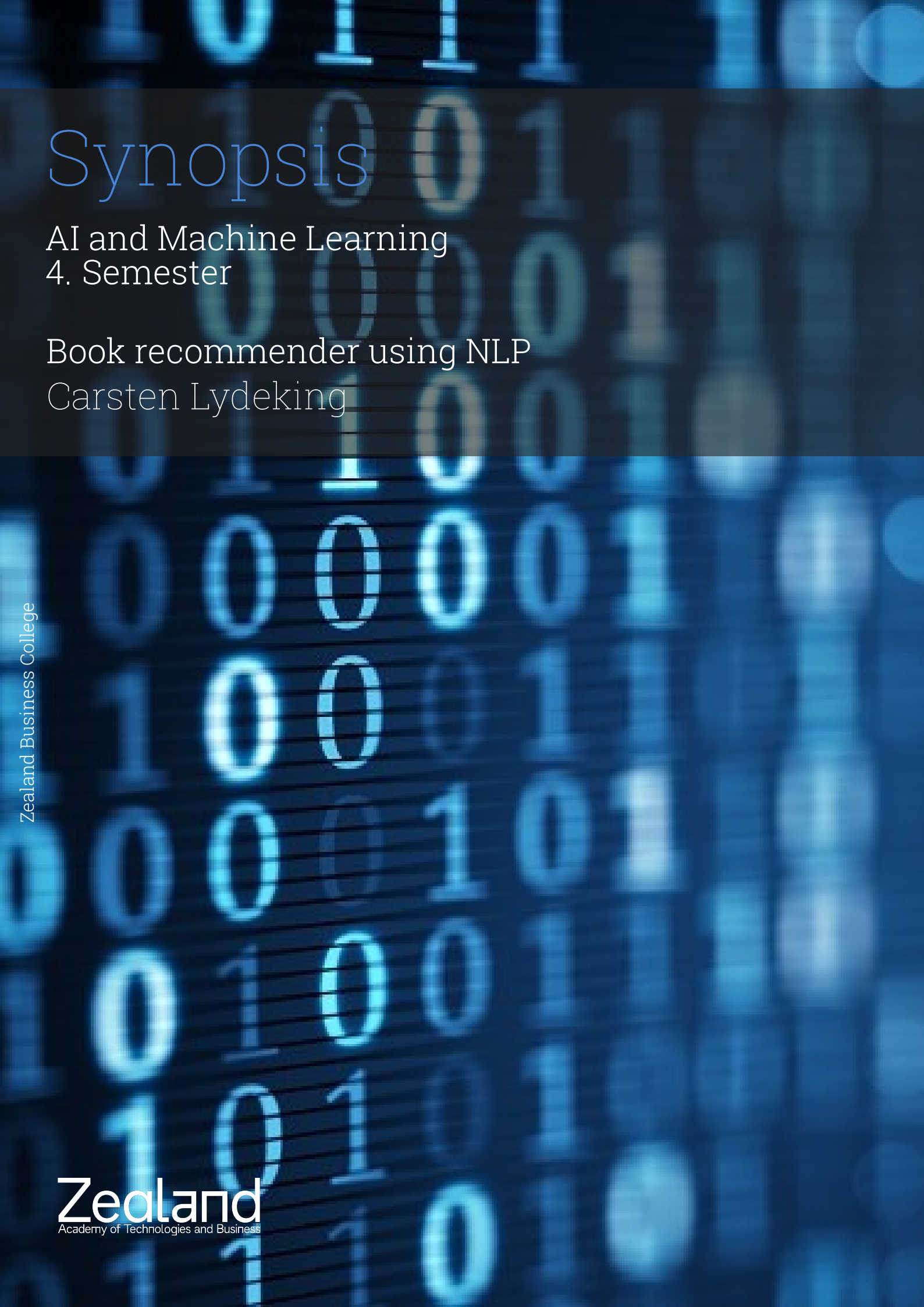


Synopsis

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. 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 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

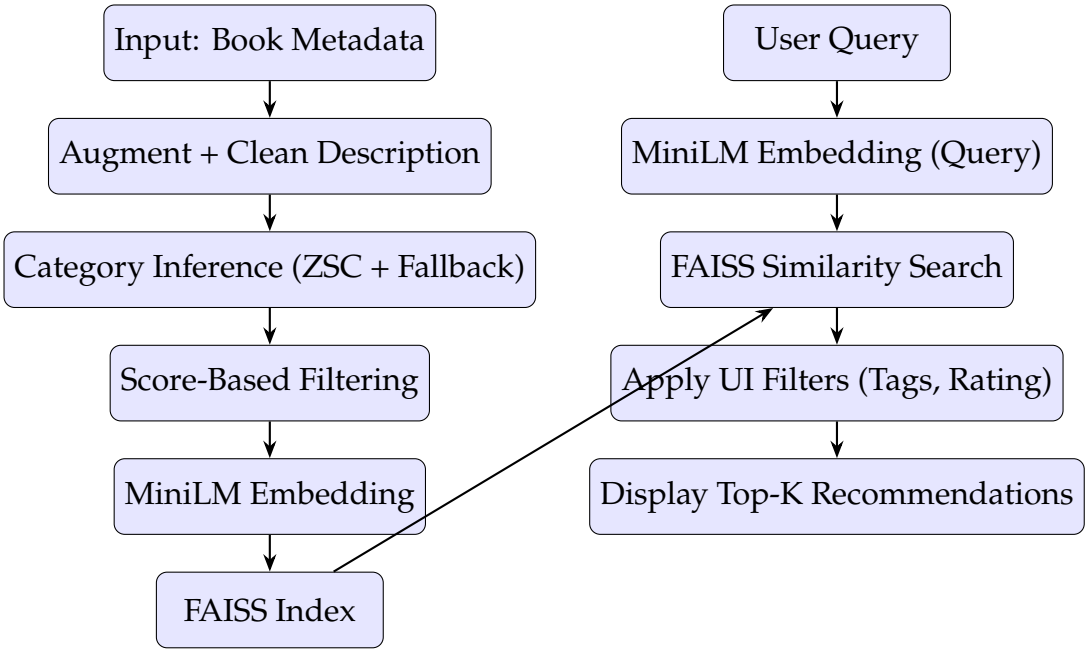
Each core system component is described in its own chapter, addressing one or more of the research sub-questions.

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 enable semantic filtering and genre-aware recommendations, each book was assigned one or more thematic categories. This process combined zero-shot classification, keyword-based fallback strategies, and confidence-based filtering.

4.1. Zero-Shot Classification with BART-MNLI

Each book's metadata was combined into an `augmented_description`, containing title, author, publication year, and description. This composite text was passed into the `facebook/bart-large-mnli` model using Hugging Face's zero-shot classification pipeline (Facebook AI, 2025).

Thirteen candidate labels were defined (e.g., *Fantasy*, *Love*, *Non-fiction*). For each label, a confidence score was returned. Only scores ≥ 0.4 were retained to ensure relevance. Books often received multiple labels.

4.2. Keyword-Based Fallback Enrichment

For books with weak model predictions or no confident labels, a curated list of keywords was used to infer likely genres. These fallback rules were only triggered if the corresponding category was not already predicted.

For example:

- **Fantasy:** *wizard*, *magic*, *dragon*
- **Science Fiction:** *cyborg*, *space travel*, *AI*
- **Love:** *romance*, *relationship*, *passion*

Fallback tagging increased recall and thematic coverage. Out of 6,572 entries, 4,665 received at least one fallback label.

4.3. Refinement and Category Merging

To reduce noise and improve label interpretability, overlapping or ambiguous categories were merged:

- *Biography + History* $\rightarrow$ *Historical*
- *Suspense + Detective* $\rightarrow$ *Mystery*
- *Poetry + Philosophy* $\rightarrow$ *Philosophy & Poetry*

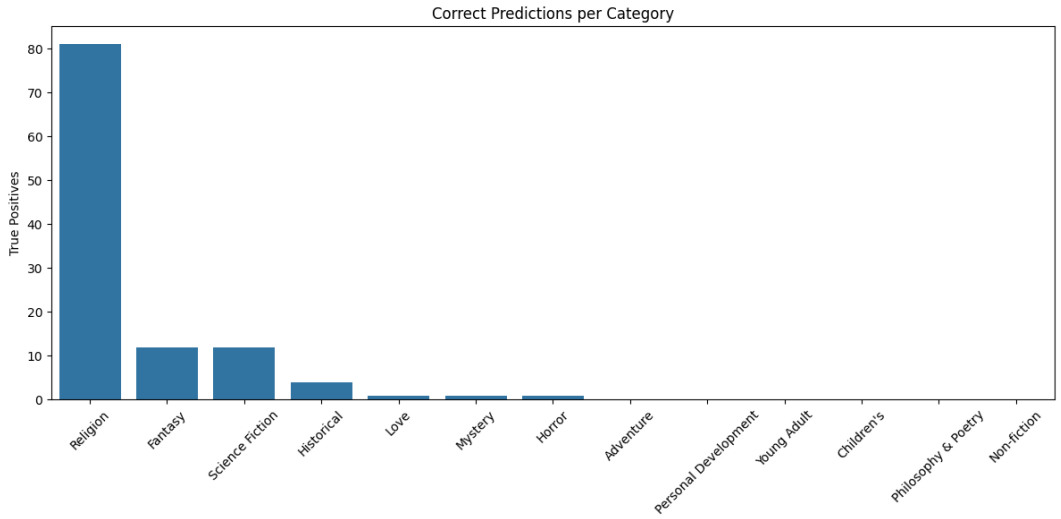


Figure 4.1: Prediction Matches per Category: Bar chart of true positives per category. Reflects model strength in domains like Fantasy and Love and highlights where fallback logic is critical.

Per-category precision, recall, and F1 scores were calculated by comparing predicted labels to a cleaned set of original tags. Figure 4.1 shows prediction matches by category.

4.4. Confidence Metrics and Filtering

To prepare the dataset for embedding and indexing, multiple metrics were computed:

- `max_score` — Highest score among all predicted labels
- `filtered_avg_score` — Mean score of predictions ≥ 0.2
- `score_std` — Standard deviation across all label scores
- `num_categories` — Count of confident labels retained

A row was retained if it met all of the following:

- `description_length` ≥ 200 characters
- `filtered_avg_score` ≥ 0.2
- `max_score` ≥ 0.4
- `num_categories` > 0

This reduced the dataset from 6,572 to 5,160 high-confidence books.

4.5. Length vs Confidence Correlation

To verify that longer descriptions produce better classifications, correlation coefficients were computed:

- Pearson $r = 0.398$, $p < 0.0001$
- Spearman $\rho = 0.261$, $p < 0.0001$

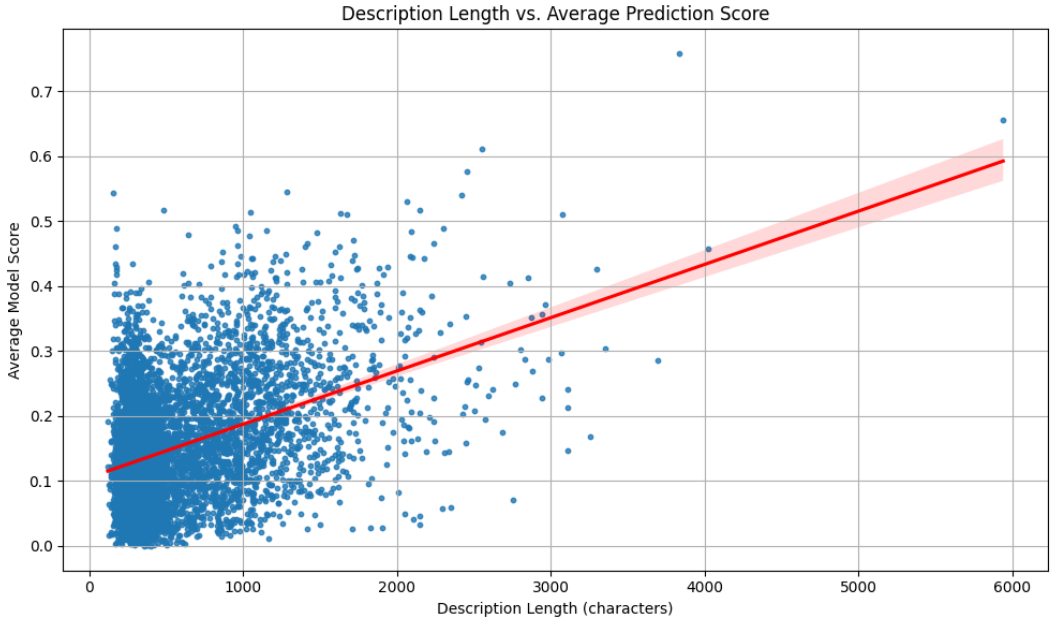


Figure 4.2: Description Length vs. Confidence Score: A regression plot showing the positive correlation between description length and average model confidence.

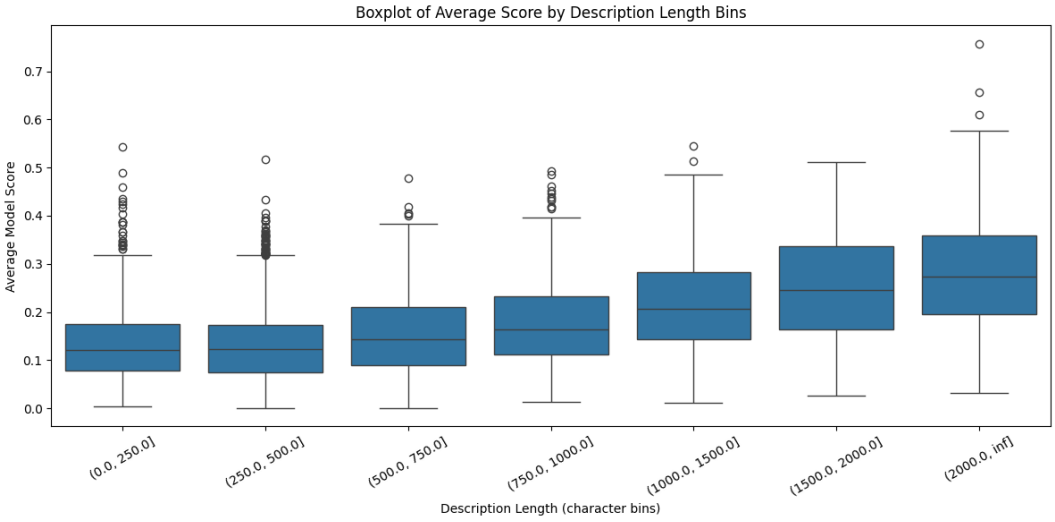


Figure 4.3: Score by Length Bin: A boxplot displaying score variation across binned description lengths.

These results indicate a moderate positive relationship. Figure 4.2 visualizes the regression, while Figure 4.3 shows variation across bins.

These findings supported a minimum `description_length` of 200 characters during filtering.

5
Text Embedding

The core of the recommendation system lies in transforming books and queries into a shared semantic space. This is accomplished by generating sentence embeddings that encode the meaning of input text into numerical vectors. Once embedded, semantic similarity can be computed between queries and books.

5.1. Sentence Embedding Theory

Sentence embeddings are dense vector representations of variable-length text sequences. They enable comparison of textual content by mapping semantically similar sentences to nearby points in vector space. The process relies on transformer-based models.

Let $f(t)$ be the embedding function applied to a string t, producing a vector $\vec{v} \in \mathbb{R}^{384}$:

$$\vec{v} = f(t), \quad \vec{v} \in \mathbb{R}^{384} \tag{5.1}$$

This project uses `all-MiniLM-L6-v2` from the Hugging Face `sentence-transformers` library (Hugging Face, 2025), which produces 384-dimensional embeddings. It offers an excellent balance between speed and semantic quality, making it ideal for offline inference on CPU hardware.

5.2. Embedding Implementation

Text embeddings were generated using a composite string field `search_text`, constructed for each book as follows:

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

This format ensures that both metadata and narrative content contribute to the semantic embedding. The following steps were performed:

1. Loaded the final dataset `books_indexed.csv`.
2. Encoded each `search_text` entry using MiniLM.
3. Saved the resulting matrix of shape $(n, 384)$ to memory.
4. Added all vectors to a FAISS index (see Chapter 6).
5. Saved metadata with vectors to `books_indexed_with_embeddings.csv`.

5.3. Embedding Configuration

The embedding stage used the following parameters:

- **Model:** `all-MiniLM-L6-v2`

- **Library:** `sentence-transformers`
- **Embedding Dimension:** 384
- **Distance Metric:** L2 (Euclidean)
- **Backend:** CPU inference
- **Input Field:** `search_text`
- **Entries Embedded:** 5,160 books

5.4. Summary

The embedding pipeline enabled semantic similarity computations between books and queries. Using a compact transformer model, each entry was encoded into a vector capturing its latent meaning. This allowed content-based querying without relying on metadata fields like genre, author popularity, or user behavior. The resulting vectors were stored in a FAISS index for efficient similarity search.

6

Vector Similarity Search with FAISS

Once books and queries are embedded into the same semantic space, the system must identify the most relevant books for a given user input. This is achieved using vector similarity search. In this project, the FAISS (Facebook AI Similarity Search) library (AI, 2025) is used to enable fast and accurate nearest-neighbor retrieval over the 5,160 book vectors.

6.1. Similarity Search Theory

Let $\vec{q} \in \mathbb{R}^{384}$ represent the embedded user query, and let $\{\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_n\}$ denote the embedded book vectors. The similarity between the query and each book is computed using squared L2 (Euclidean) distance:

$$\text{dist}(\vec{q}, \vec{b}_i) = \|\vec{q} - \vec{b}_i\|_2^2 \tag{6.1}$$

The top k nearest neighbors (i.e., those with the smallest distances) are returned as recommended books. This approach assumes that semantically similar texts lie close together in vector space.

6.2. Why FAISS?

FAISS is optimized for fast similarity search over high-dimensional vectors. It was selected for this project because it:

- **Performs well on CPU** — fast enough for real-time use
- **Supports exact and approximate search** — adaptable for scale
- **Offers simple integration with NumPy and PyTorch**
- **Runs offline** — no internet connection required

This implementation uses `IndexFlatL2`, a brute-force index that performs exact nearest-neighbor search using squared L2 distance. It is suitable for small to medium datasets and does not require tuning.

6.3. Implementation

After the embeddings were generated (see Chapter 5), the similarity search process was implemented as follows:

1. Initialize a `faiss.IndexFlatL2` index with vector dimension 384.
2. Add the 5,160 book vectors to the index.

3. At runtime:
    - Embed the user query using the same MiniLM model.
    - Use FAISS to find the top k nearest book vectors.
    - Retrieve metadata from `books_indexed_with_embeddings.csv`.

The system achieves sub-second response times on consumer-grade laptops using only CPU.

6.4. Summary

FAISS powers the core similarity engine of the recommender system. Combined with MiniLM embeddings, it enables local, real-time, and semantically aware recommendations. The entire matching process relies solely on content, requiring no user profiles or interaction history.

7

User Interface

The user interface enables interaction with the semantic book recommendation system. While not a machine learning component, it plays a crucial role in usability, filtering, and presentation. The interface is implemented using `Streamlit`, a lightweight Python framework suitable for local-first applications.

7.1. Design Principles

The UI was designed with simplicity and responsiveness in mind. It supports end-to-end offline operation. Design goals included:

- Concept-based natural language search
- Multi-faceted filtering by category and rating
- Minimal dependencies and fast startup
- Clear presentation of recommendations

All inference and retrieval logic runs locally.

7.2. Query Workflow

The system accepts a free-form text query from the user (e.g., *"Books about surviving on Mars"*). The workflow proceeds as follows:

1. Query is embedded using the MiniLM model.
2. The FAISS index returns the top 60 nearest book vectors.
3. Filtering and sorting options are applied.
4. Six book cards are rendered per page.

This approach allows expressive, language-based search.

7.3. Filtering and Sorting

Users can refine results with:

- **Category filter:** Multi-select widget for genre labels
- **Rating sort:** Toggle for ascending/descending order
- **Pagination:** Adjustable page selection for navigation

Filters are applied after FAISS similarity search.

7.4. Result Display

Each result is rendered as a card containing:

- Book cover image (URL with fallback)
- Full title and author
- Average rating, number of pages, and publication year
- Description excerpt (up to 300 characters)
- Refined categories

The layout ensures readability and intuitive navigation.

7.5. Privacy and Offline Execution

The application is designed for full offline use:

- No tracking, cookies, or login
- No data transmission to external servers
- Book data, models, and index are stored locally

This ensures data privacy, fast load times, and compatibility with low-resource environments.

7.6. Summary

The Streamlit interface allows users to interact with the system seamlessly. Natural language search, combined with real-time filtering and semantic recommendations, creates an intuitive and private discovery experience.

8

Performance and Evaluation Challenges

This chapter outlines how the system's performance was evaluated, given the absence of user feedback and labeled test data. Traditional metrics such as accuracy or precision are not directly applicable to semantic similarity tasks without ground truth. Therefore, evaluation is divided into subjective, statistical, and system-level components.

8.1. Evaluation Without Labels

The system does not predict a fixed label or class. Instead, it embeds both queries and books in a shared vector space and retrieves nearest neighbors using semantic distance. As a result:

- There is no single "correct" recommendation per query.
- Precision/recall-based metrics cannot be used reliably.
- Feedback mechanisms like click-through or engagement data are unavailable in an offline setting.

This necessitates alternative evaluation strategies.

8.2. Qualitative Evaluation

Evaluation relied on exploratory queries and manual inspection of results. Test queries covered a range of genres, tones, and abstraction levels. Examples include:

- "Existential loneliness in space"
- "Post-apocalyptic survival story"
- "Books that explore grief through fantasy"

Results were judged based on alignment between the query and the content of the returned book descriptions. This human-in-the-loop approach confirmed that semantically aligned results were typically returned.

8.3. Category Inference Evaluation

Although the original dataset lacked consistent genre labels, the quality of inferred categories was assessed via:

- Retaining classification confidence scores per label
- Filtering by a threshold of 0.4 (see Chapter 4)
- Computing precision, recall, and F1 scores for a subset of trusted entries
- Visualizing match counts and fallback coverage

Certain categories (e.g., *Fantasy*, *Love*) were strongly predicted by the model, while others (e.g., *Philosophy & Poetry*) relied more heavily on fallback keyword enrichment.

8.4. Responsiveness and Latency

Performance was measured on a consumer-grade laptop (no GPU):

- Query embedding time (MiniLM): < 0.2 seconds
- FAISS top-60 vector search: < 10 ms
- Streamlit UI rendering (6 cards per page): < 2 seconds including image fallback

This confirms that the system operates in real time on modest hardware, supporting its offline-first goal.

8.5. Scalability Considerations

While the current dataset (5,160 entries) is manageable, scaling raises practical challenges:

- **Indexing:** FAISS IndexFlatL2 is efficient but linear in search time; approximate methods may be needed for larger corpora.
- **Re-embedding:** Changing models or descriptions requires recomputing all embeddings.
- **Filtering:** UI-side filtering and sorting scale linearly with post-search result volume.

Future extensions should consider vector compression, batch processing, and caching.

8.6. Limitations of Offline Evaluation

Offline evaluation limits the types of insights that can be gathered:

- **Relevance:** Relies solely on developer judgment
- **Discovery:** Cannot measure novelty or serendipity
- **Bias Detection:** Overrepresentation of certain authors or genres may go unnoticed

These are acceptable trade-offs in the context of a local, user-respecting prototype.

8.7. Summary

Evaluation focused on semantic quality, practical responsiveness, and offline usability. Despite the lack of supervised benchmarks, confidence filtering, manual validation, and runtime measurements demonstrated that the system is performant and aligned with its design goals.

9 Conclusion

This chapter summarizes the findings and outcomes of the project, reflects on its limitations, and revisits the original research questions.

9.1. Discussion

This project demonstrated that transformer-based models can power a local, content-only book recommendation system. Without requiring user profiles or collaborative filtering, the system delivers semantically meaningful recommendations based entirely on book metadata and free-text queries.

The end-to-end system consisted of:

- Augmented and cleaned metadata for over 6,800 books
- Zero-shot genre inference using BART-MNLI with fallback keyword enrichment
- Semantic embeddings via MiniLM
- FAISS-based vector search
- A responsive, private UI implemented in Streamlit

With no reliance on internet access, the system runs entirely offline and respects user privacy. It is suitable for low-resource environments and educational use cases.

9.2. Conclusion

The system successfully addressed all four research sub-questions:

1. **What techniques exist for embedding text into meaningful vectors?** Pretrained sentence transformers like MiniLM encode text into high-dimensional semantic vectors, enabling rich similarity comparisons.
2. **How can vector similarity be used for finding similar books?** FAISS enables fast nearest-neighbor retrieval in vector space, allowing queries to return top-matching books based on content alone.
3. **How can genre categories be inferred and used for filtering?** BART-MNLI enables zero-shot classification of genre labels. Fallback keyword rules and confidence filtering improve label coverage and reliability.
4. **What are the limitations of a local, content-only recommender?** The system lacks personalization and behavior-driven refinement. However, it compensates with transparency, offline execution, and semantic precision.

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 was answered by developing and validating a fully local recommendation system that performs semantic classification, embedding, indexing, and retrieval — all on-device.

9.3. Reflection

The project offered valuable insights into building ML-powered applications under strict resource constraints. Key lessons include:

- **Description quality drives results:** Well-written descriptions significantly enhance semantic accuracy.
- **Fallbacks expand coverage:** Keyword-based strategies improve classification when the model lacks confidence.
- **Local-first ML is viable:** Even compact transformer models provide robust performance on consumer hardware.
- **Simplicity over complexity:** Lightweight, modular design reduced failure points and increased maintainability.

With more time, the system could be extended to include:

- Comparison of embedding models (e.g., MPNet, SBERT)
- Re-ranking of results based on metadata
- Interactive user feedback or offline learning

The project demonstrates that privacy-respecting, offline-first NLP systems are not only possible but practical. This aligns with emerging interest in edge-based AI and user-sovereign computing.

References

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

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

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 Terms

This appendix defines key terms used throughout the project, with a focus on Machine Learning, Natural Language Processing, and Information Retrieval. Each entry aims to explain the term in accessible language, suitable for readers with a general computer science background.

AI (Artificial Intelligence): The simulation of human intelligence in machines that can reason, learn, and make decisions.

Augmentation (Data): The process of enriching data with external sources to fill in missing values or increase its descriptive quality.

BART (Bidirectional and Auto-Regressive Transformer): A transformer model used for text generation and classification tasks. In this project, it powers zero-shot classification.

Category Inference: Assigning thematic or genre labels to text entries based on their content.

Content-Based Recommendation: A technique that recommends items based on their attributes rather than user behavior.

Embedding: The transformation of text into a numeric vector that captures its meaning in a high-dimensional space.

FAISS (Facebook AI Similarity Search): A library that enables efficient vector similarity search, often used for large-scale nearest neighbor search.

Filtering: Reducing a set of results based on certain constraints (e.g., rating threshold, genre tag).

Inference (Model): The process of using a trained model to make predictions on new, unseen data.

L2 Distance (Euclidean): A metric that measures the straight-line distance between two vectors in space.

Local-First Design: A system architecture that prioritizes local computation and storage, minimizing reliance on external servers.

MiniLM: A lightweight sentence embedding model that provides efficient semantic representations with low computational cost.

Query: A user-inputted string used to search or retrieve information from a system.

Recommender System: A system that suggests items of interest to users based on various algorithms.

Semantic Search: A search technique that considers the meaning of the query rather than just keyword matching.

Sentence Embedding: A vector representation of a sentence that captures its semantic meaning.

Streamlit: A lightweight Python framework for building interactive web interfaces for machine learning and data science applications.

Transformer: A neural network architecture designed to handle sequential data, commonly used in language models.

Vector Index: A data structure that stores embedded vectors for fast similarity retrieval.

Zero-Shot Classification: A method where a model assigns labels it has never seen during training, using semantic understanding.

B

Symbol Dictionary

This appendix summarizes the mathematical symbols used in the project. Each symbol is listed with its meaning and the context in which it appears.

t Input text or string to be embedded (e.g., book description or user query).

$f(t)$ Embedding function that maps a string to a high-dimensional vector using a pretrained model.

$\vec{v}$ The resulting vector embedding of the input text.

$\mathbb{R}^d$ The d-dimensional real-valued vector space in which embeddings are located. In this project, $d = 384$.

$\vec{q}$ The vector representation of a user query.

$\vec{b}_i$ The vector representation of the i-th book in the dataset.

$\|\vec{q} - \vec{b}_i\|_2^2$ The squared L2 (Euclidean) distance between the query and a book vector, used for similarity ranking.

k The number of top search results to retrieve based on vector similarity.

n Total number of book entries in the vector index.

r Pearson correlation coefficient, used to measure the linear relationship between description length and confidence.

ρ Spearman rank correlation coefficient, used to evaluate monotonic relationships between two ranked variables.

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.

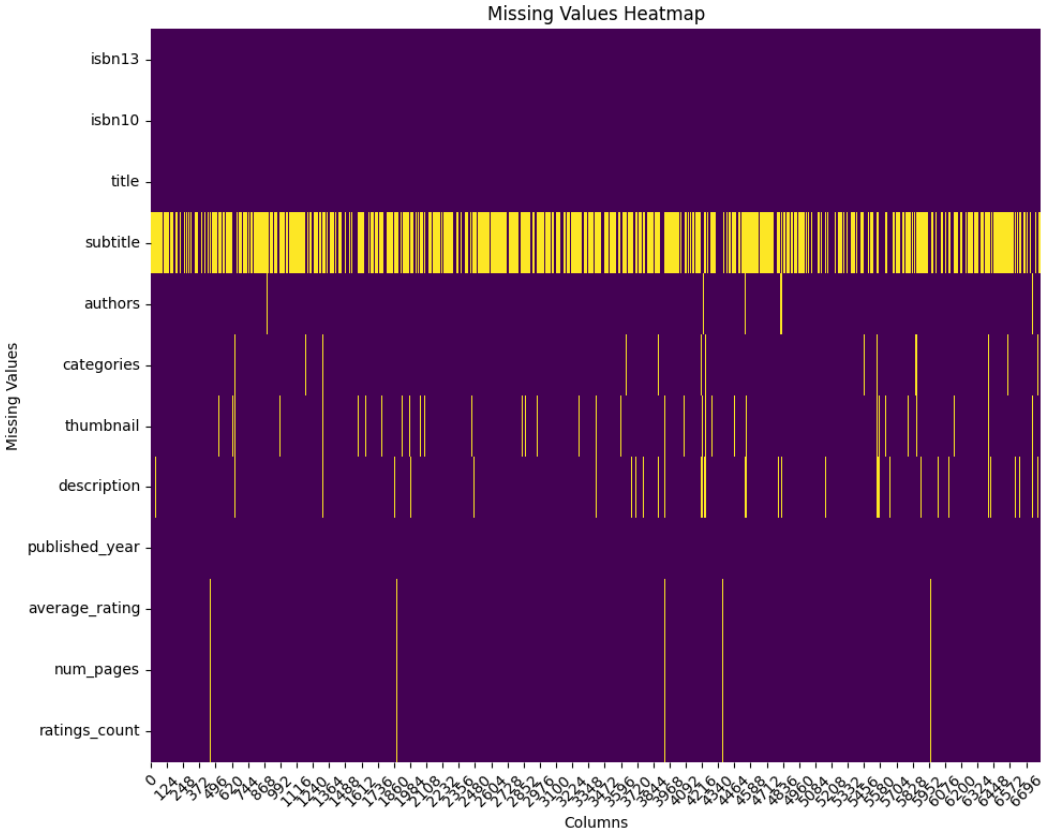


Figure C.1: Missing Values Heatmap: Visualizes the density of missing fields in the original dataset. The heatmap highlights common gaps such as missing subtitles, authors, and thumbnails, prompting the need for augmentation.

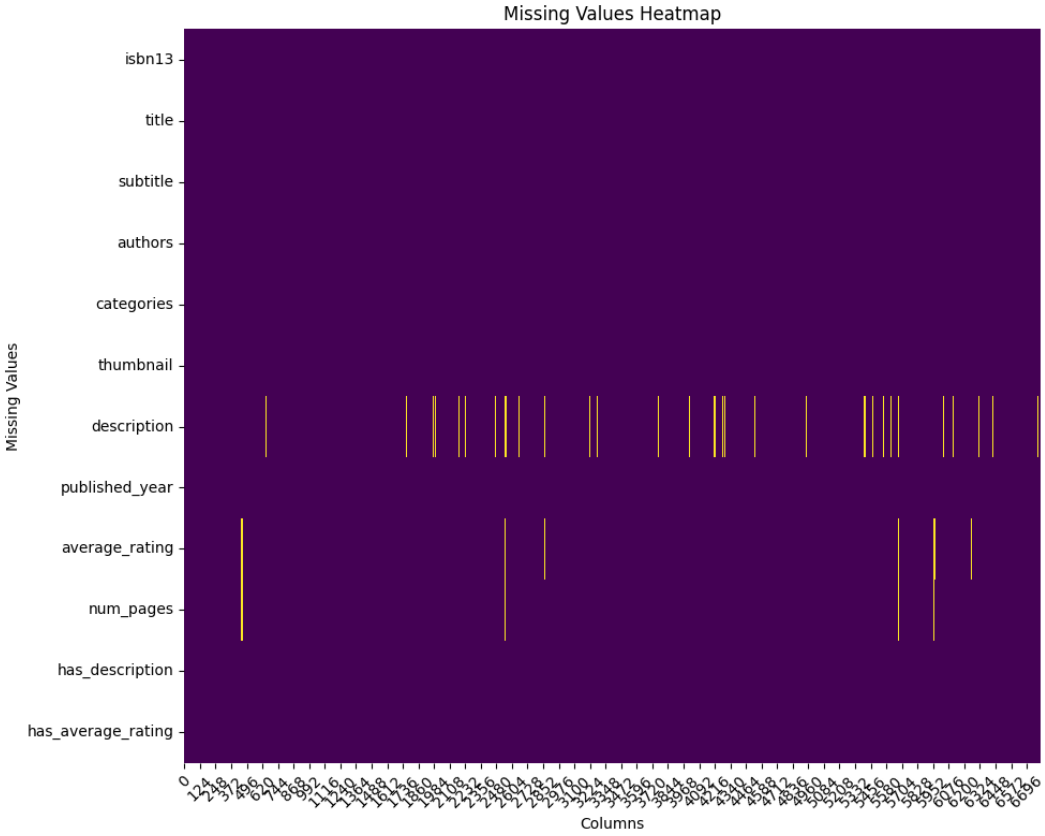


Figure C.2: OpenLibrary Completion Heatmap: Displays the remaining missing values after OpenLibrary augmentation. It shows improvements in coverage of fields like subtitles, page counts, and published years.

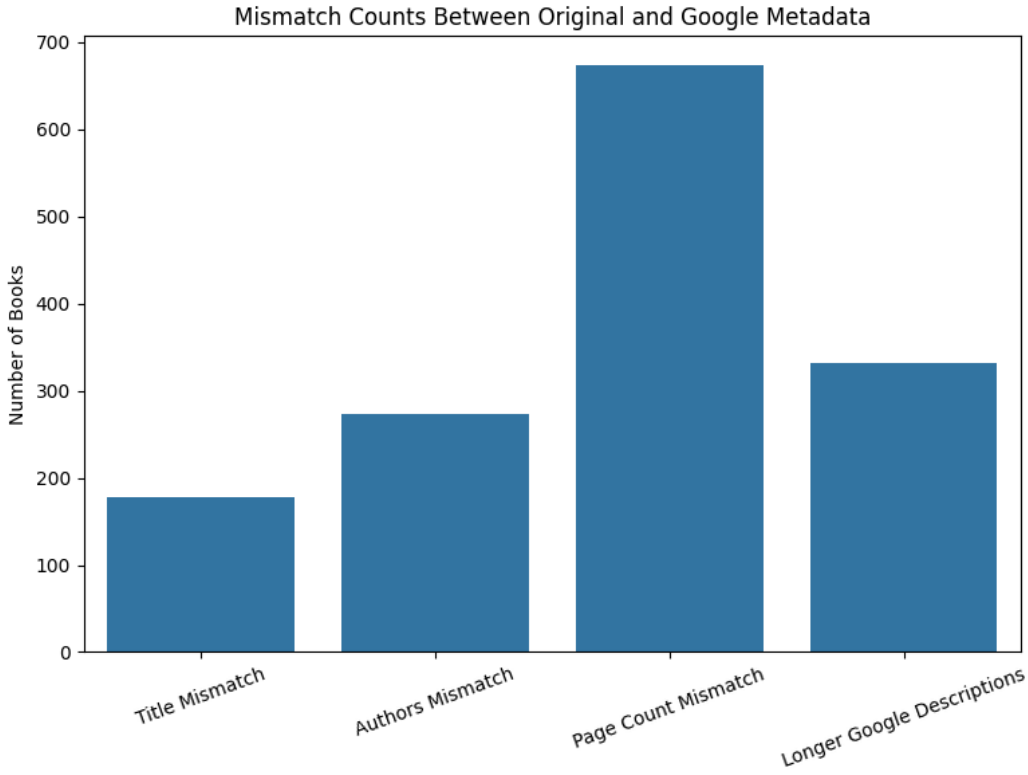


Figure C.3: Metadata Conflict Counts: A bar chart showing the number of books with mismatches in title, author, or page count across sources. This justified creation of alternative metadata fields.

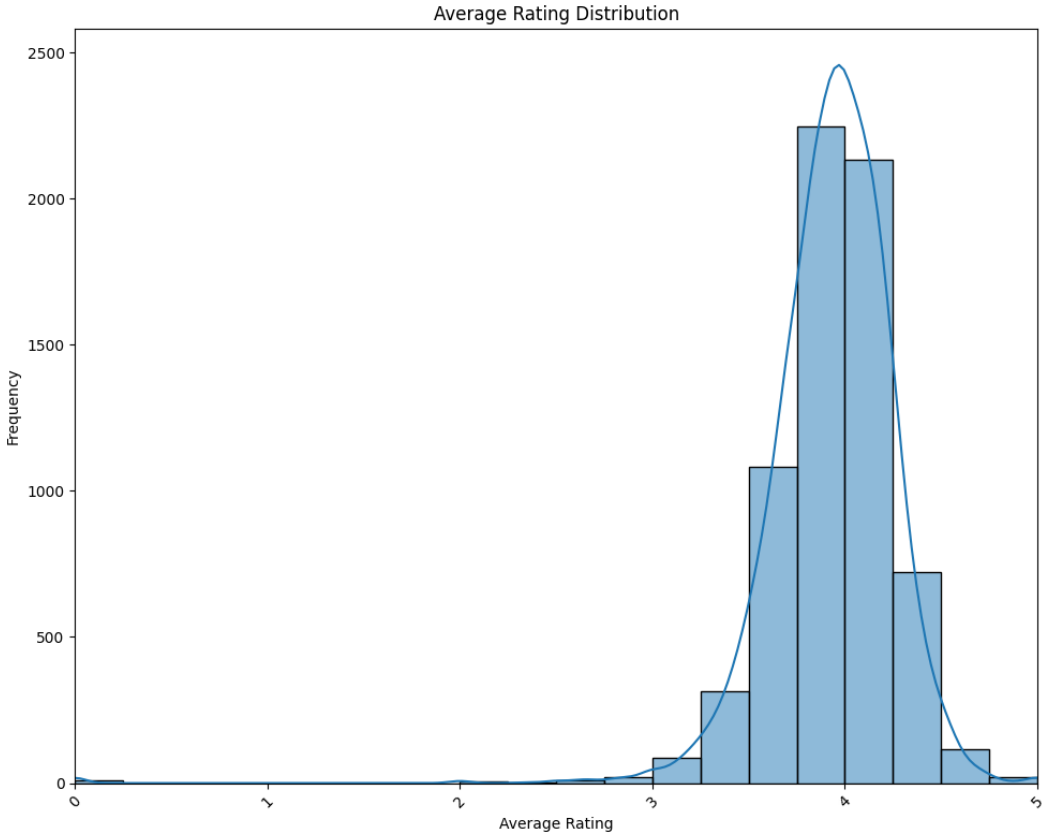


Figure C.4: Rating Distribution: Histogram of average book ratings. The skew toward higher ratings is typical for user-generated content and informed filter default settings.

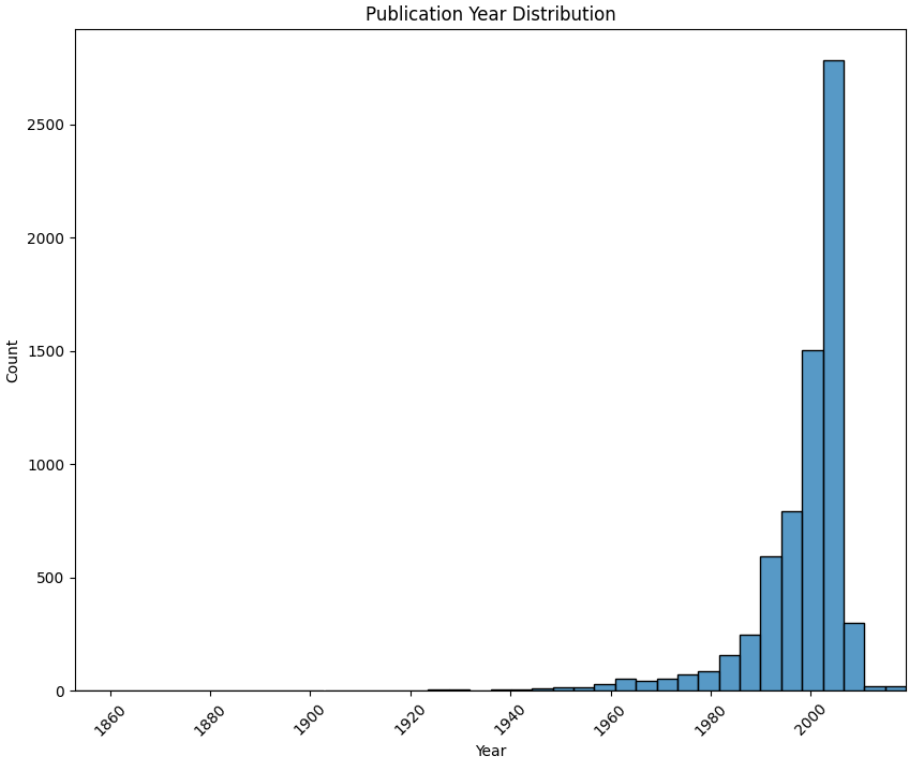


Figure C.5: Publication Year Distribution: Shows the number of books published per year. The majority are post-2000, indicating a recency bias that may impact topic coverage.

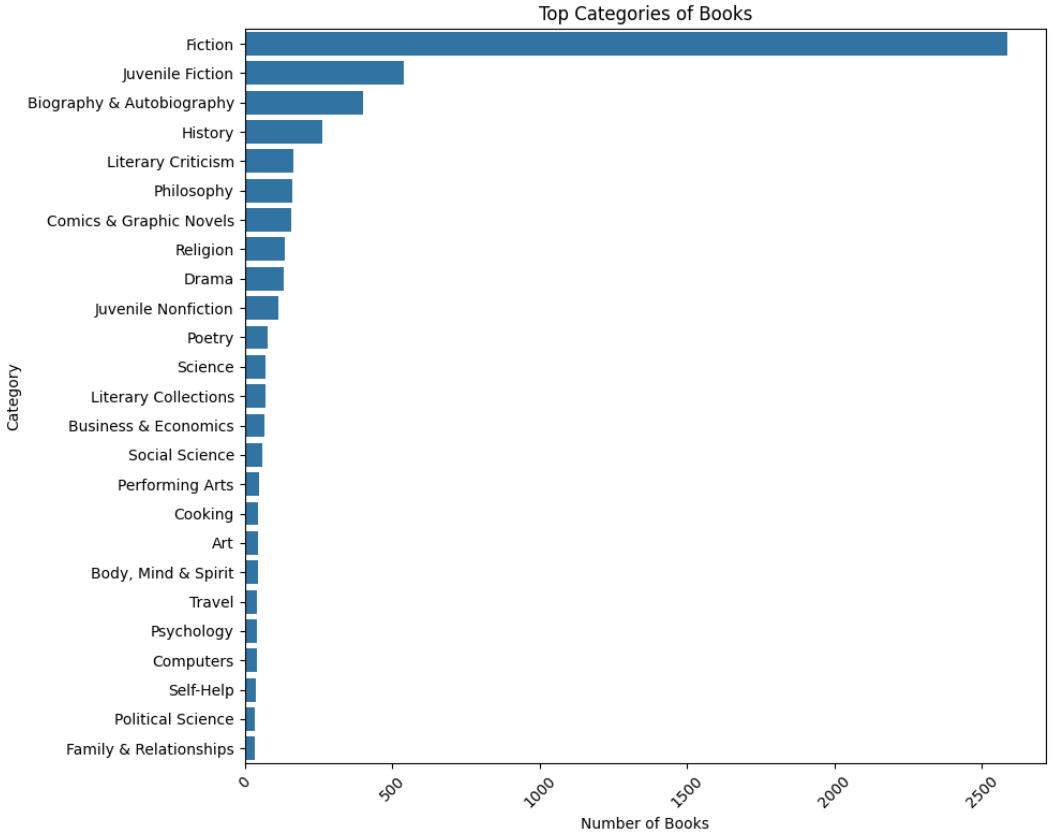


Figure C.6: Top Categories: A bar chart of the most frequent category labels in the original dataset. This informed the definition of candidate labels for zero-shot classification.

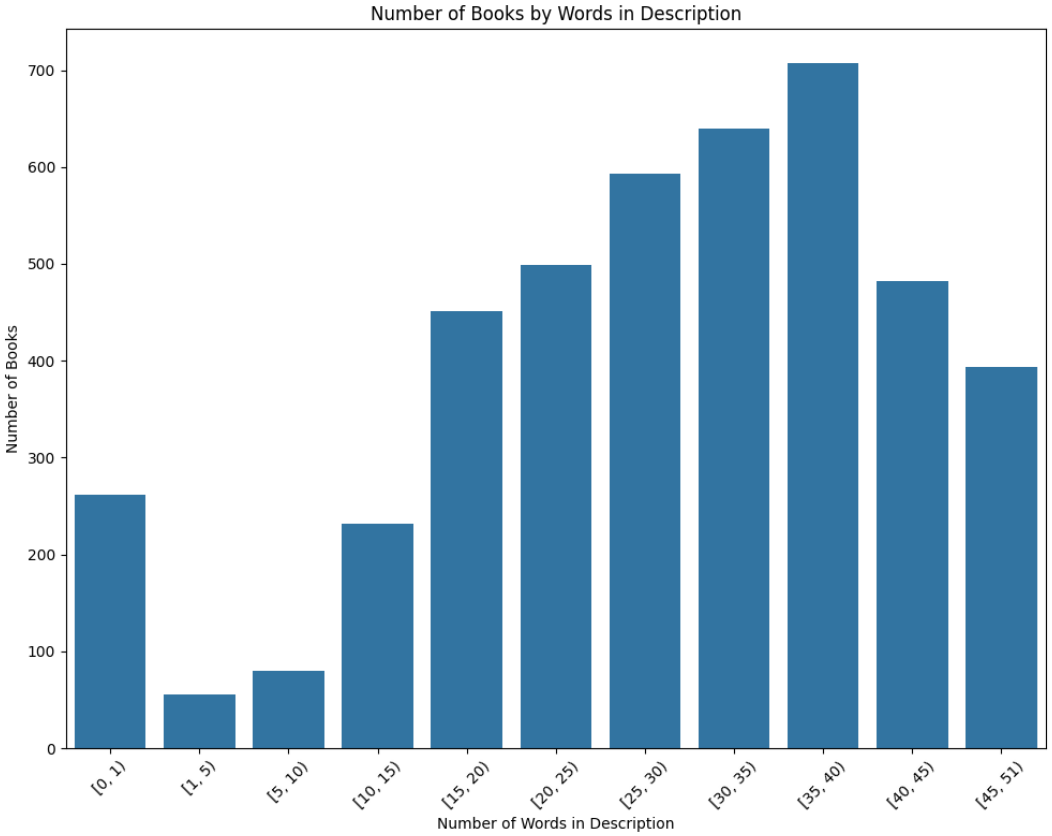


Figure C.7: Books with Short Descriptions: Distribution of books with fewer than 50 words in their descriptions. These entries were flagged for enrichment or removal.

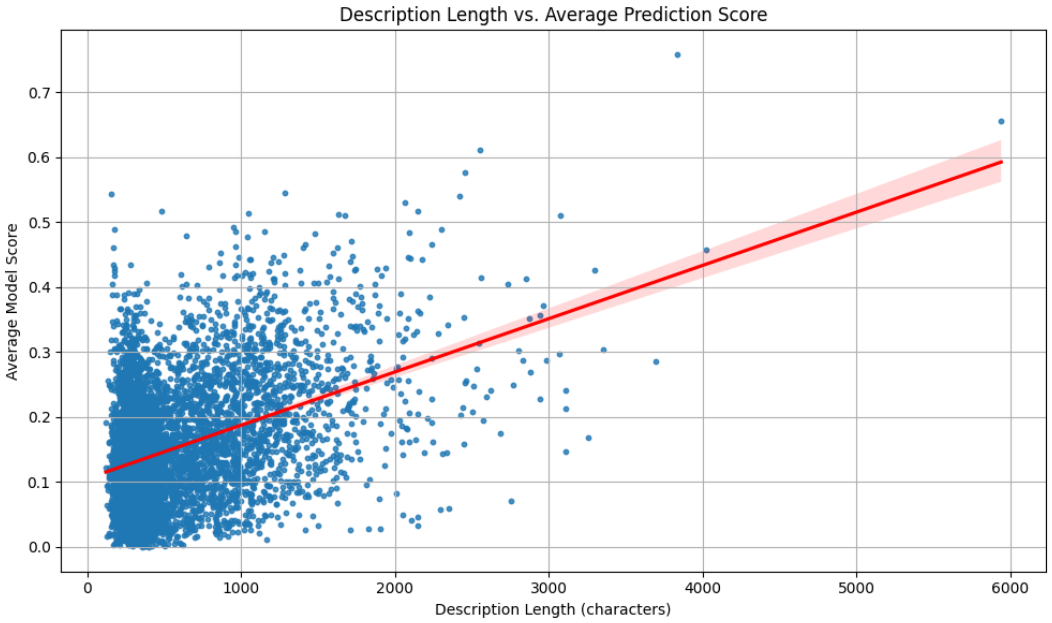


Figure C.8: Description Length vs. Confidence Score: A regression plot showing the positive correlation between description length and average model confidence. This supported the threshold of 200 characters.

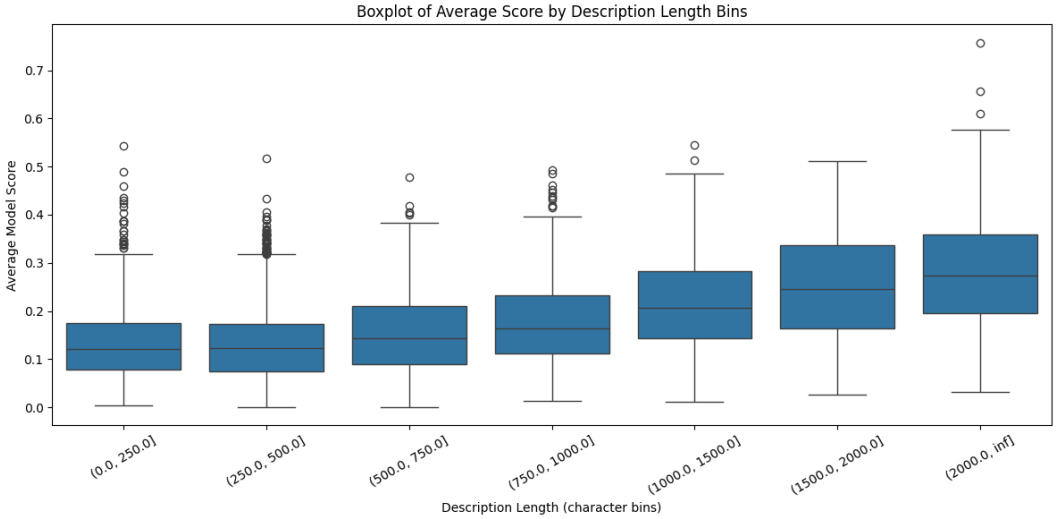


Figure C.9: Score by Length Bin: A boxplot displaying score variation across binned description lengths. Helps visualize consistency and reliability of model outputs by text richness.

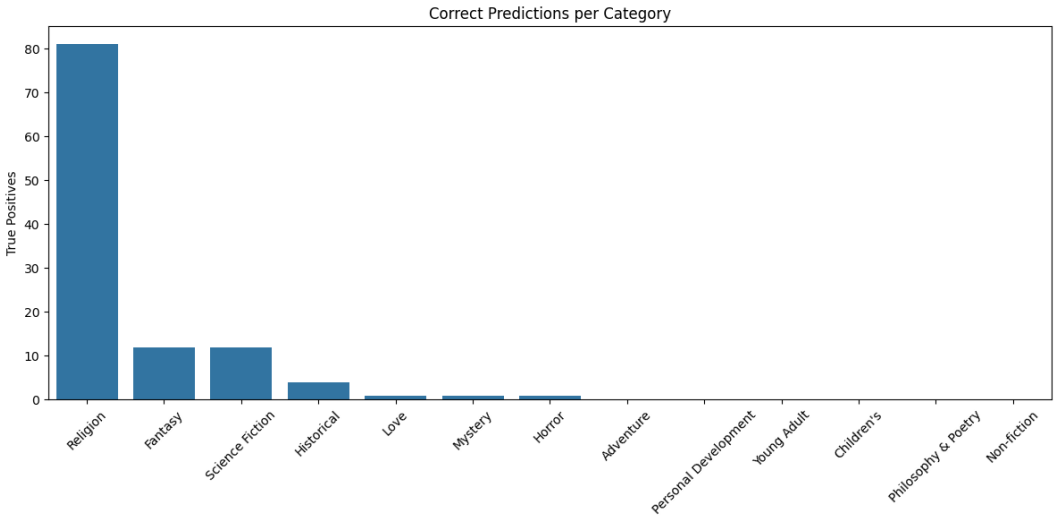


Figure C.10: Prediction Matches per Category: Bar chart of true positives per category. Reflects model strength in domains like Fantasy and Love and highlights where fallback logic is critical.

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.