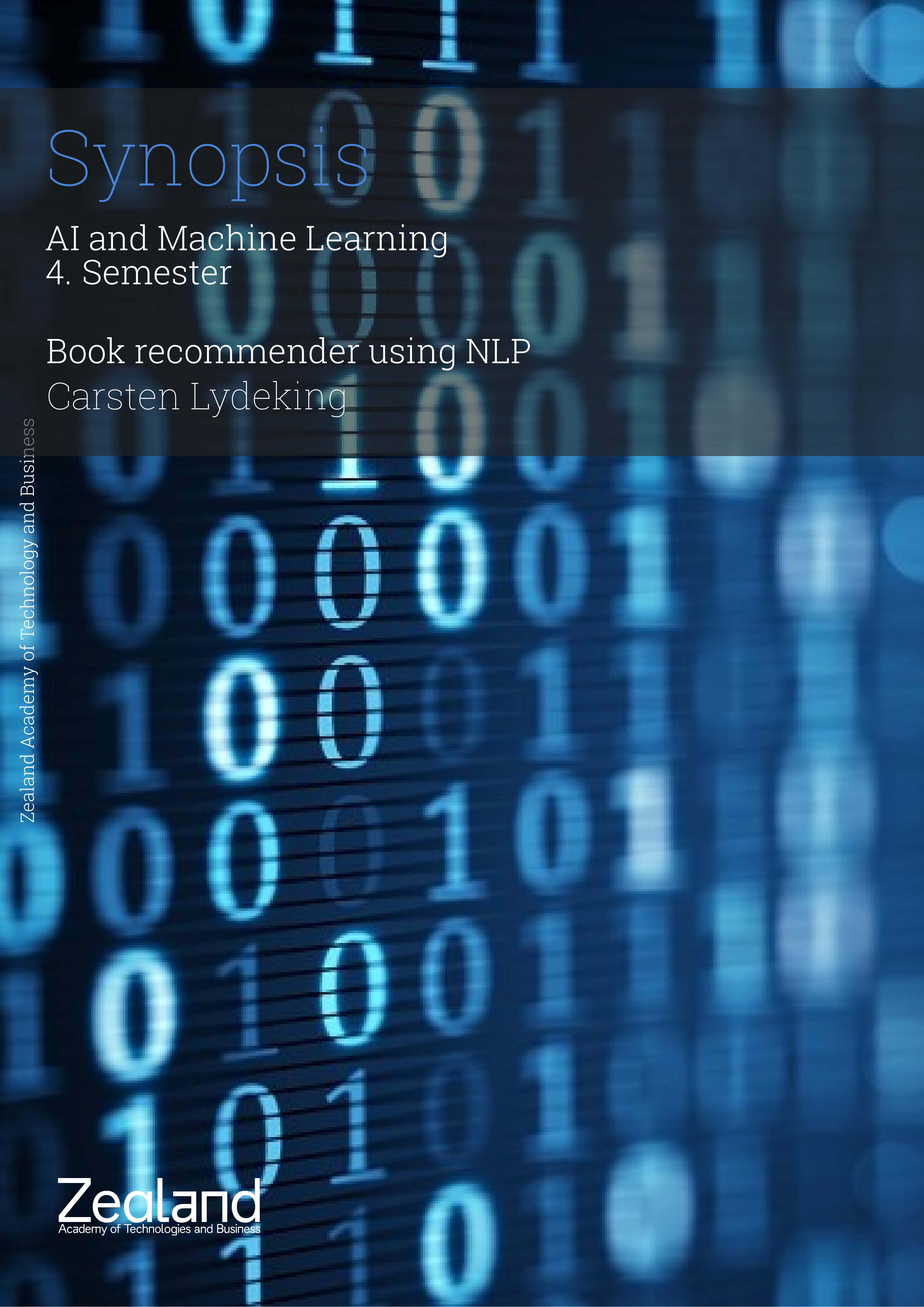


Synopsis

AI and Machine Learning
4. Semester

Book recommender using NLP

Carsten Lydeking

Zealand Academy of Technology and Business

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:	28/05/2025
Handed-in:	28/05/25
Semester:	4. Semester
Word Count:	23.460 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

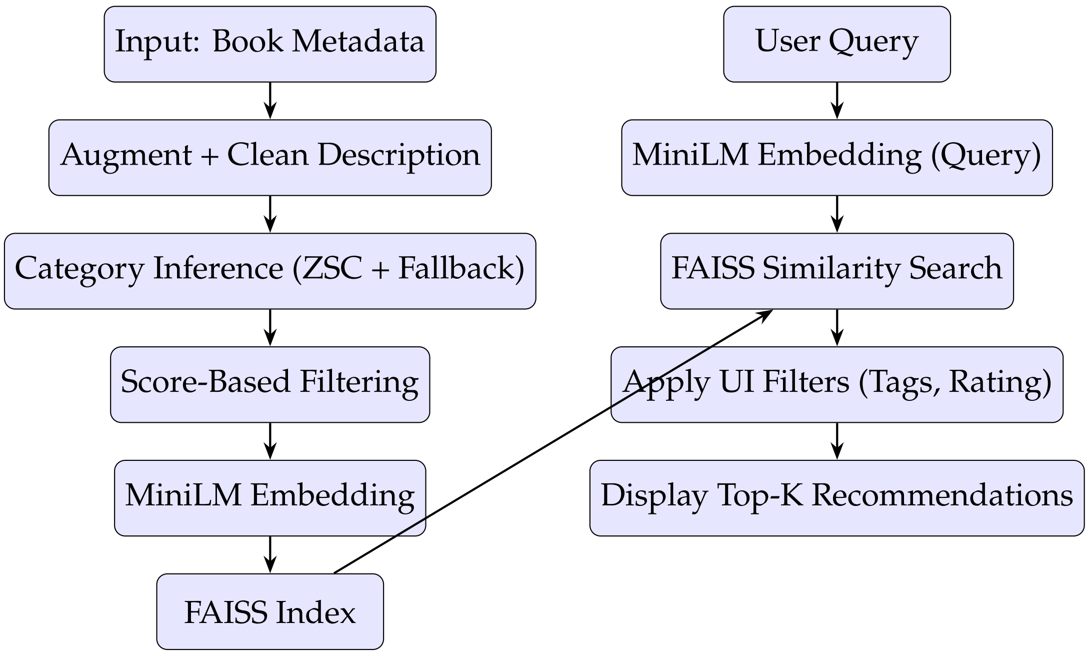
Each core system component is described in its own chapter, addressing one or more of the research sub-questions.

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* $\longrightarrow$ *Historical*
- *Suspense + Detective* $\longrightarrow$ *Mystery*
- *Poetry + Philosophy* $\longrightarrow$ *Philosophy & Poetry*

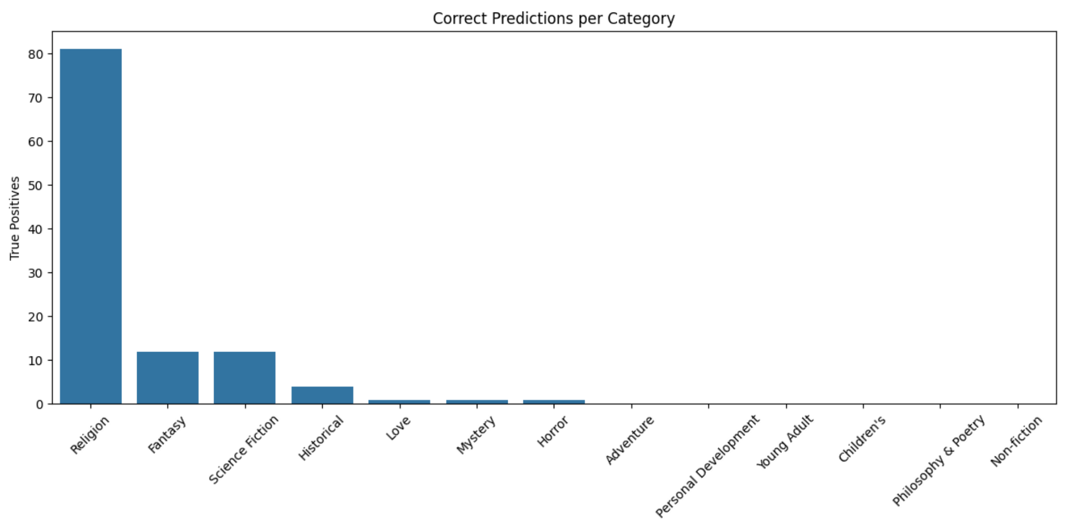


Figure 4.1: Prediction Matches per Category: Bar chart of true positives per category. Reflects model strength in domains like Fantasy and Love and highlights where fallback logic is critical.

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 $\rho = 0.261$, $p < 0.0001$

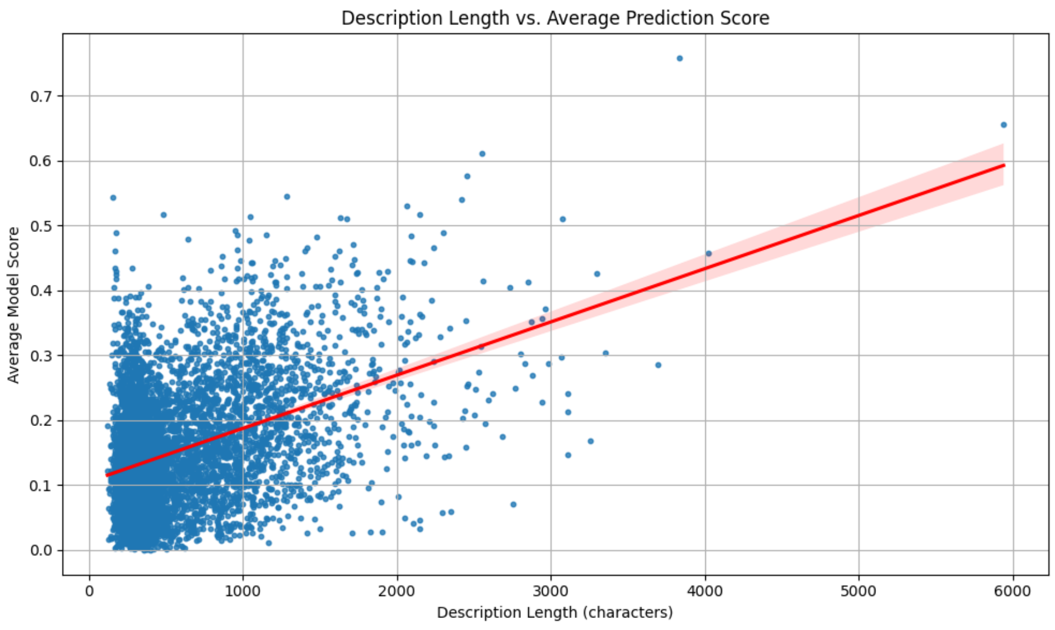


Figure 4.2: Description Length vs. Confidence Score: A regression plot showing the positive correlation between description length and average model confidence.

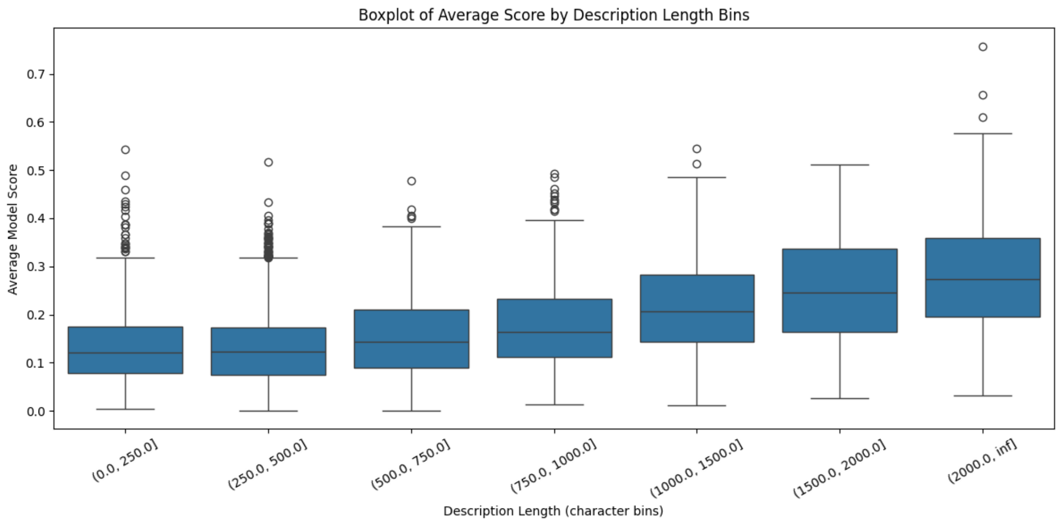


Figure 4.3: Score by Length Bin: A boxplot displaying score variation across binned description lengths.

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

Johnson, J., Douze, M., & Jégou, H. (2021). Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, *7*(3), 535–547.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543.

Reimers, N., & Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 3982–3992.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *30*.

Williams, A., Nangia, N., & Bowman, S. R. (2018). A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

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.

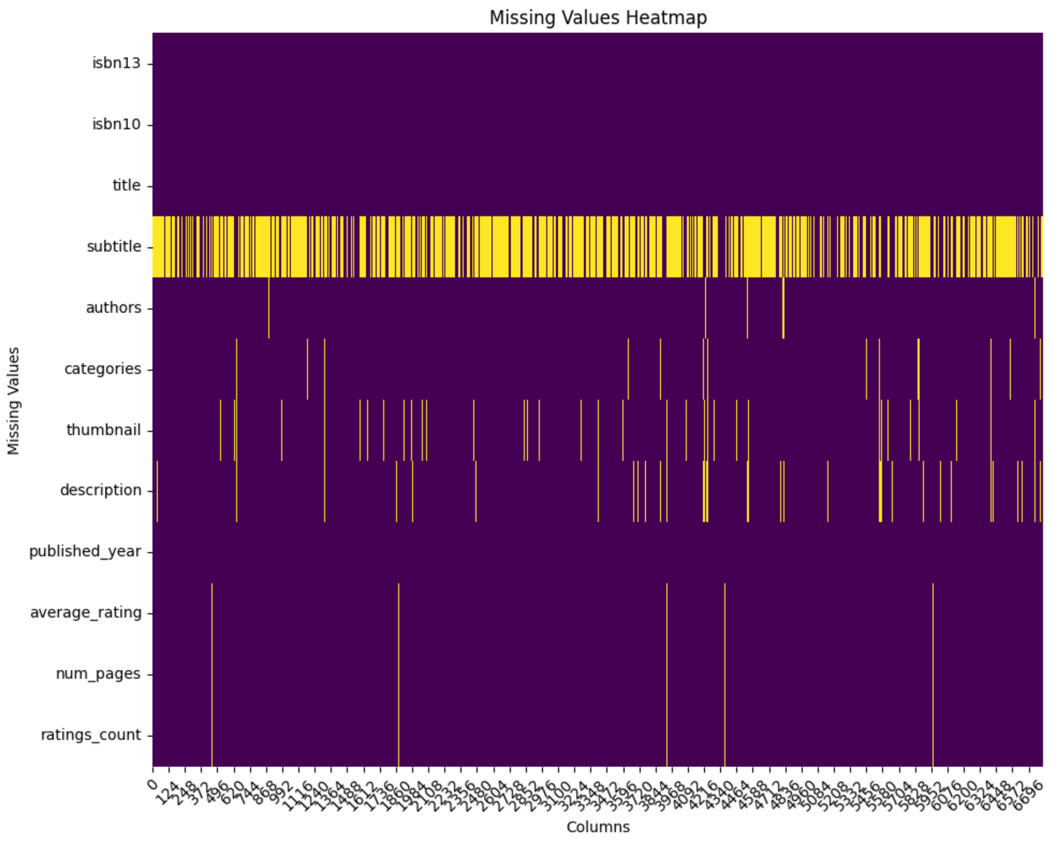


Figure C.1: Missing Values Heatmap: Visualizes the density of missing fields in the original dataset. The heatmap highlights common gaps such as missing subtitles, authors, and thumbnails, prompting the need for augmentation.

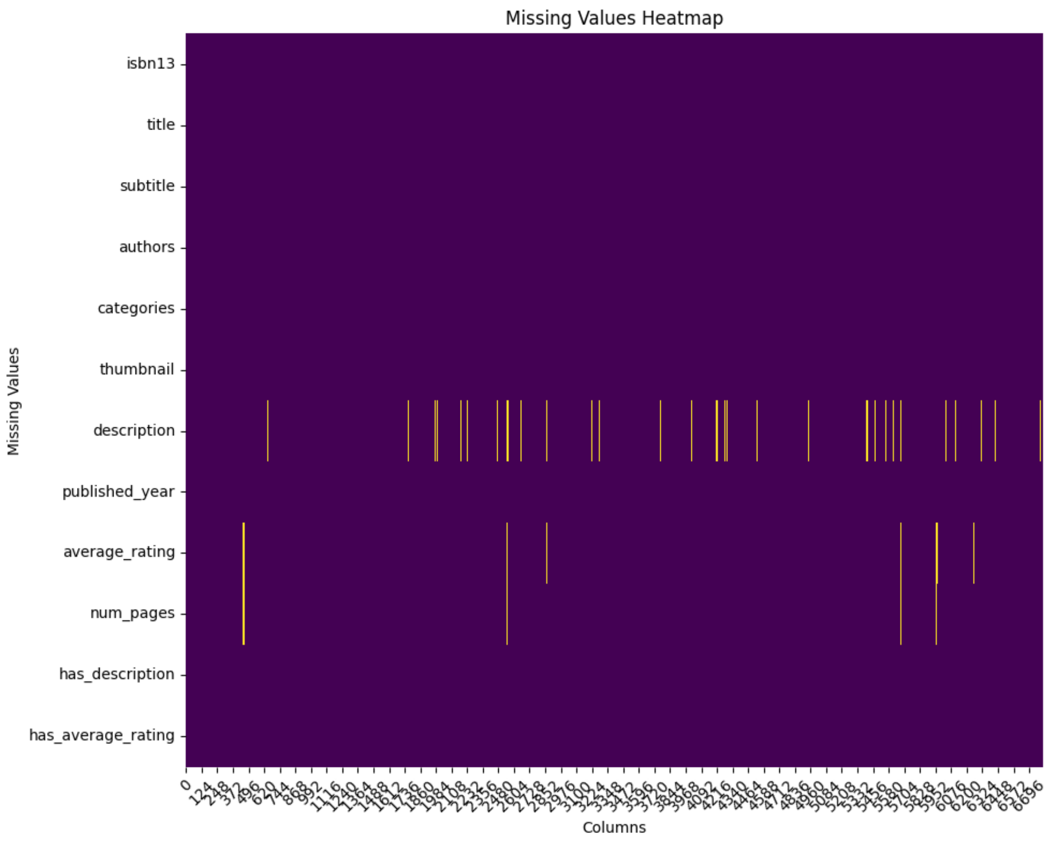


Figure C.2: OpenLibrary Completion Heatmap: Displays the remaining missing values after OpenLibrary augmentation. It shows improvements in coverage of fields like subtitles, page counts, and published years.

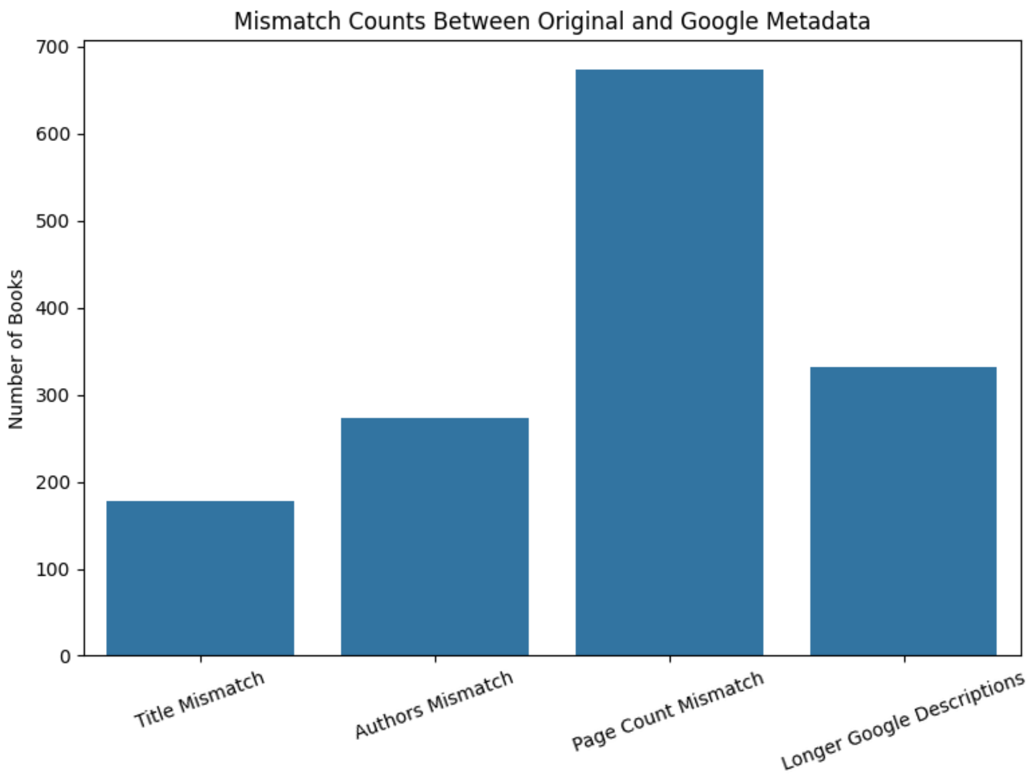


Figure C.3: Metadata Conflict Counts: A bar chart showing the number of books with mismatches in title, author, or page count across sources. This justified creation of alternative metadata fields.

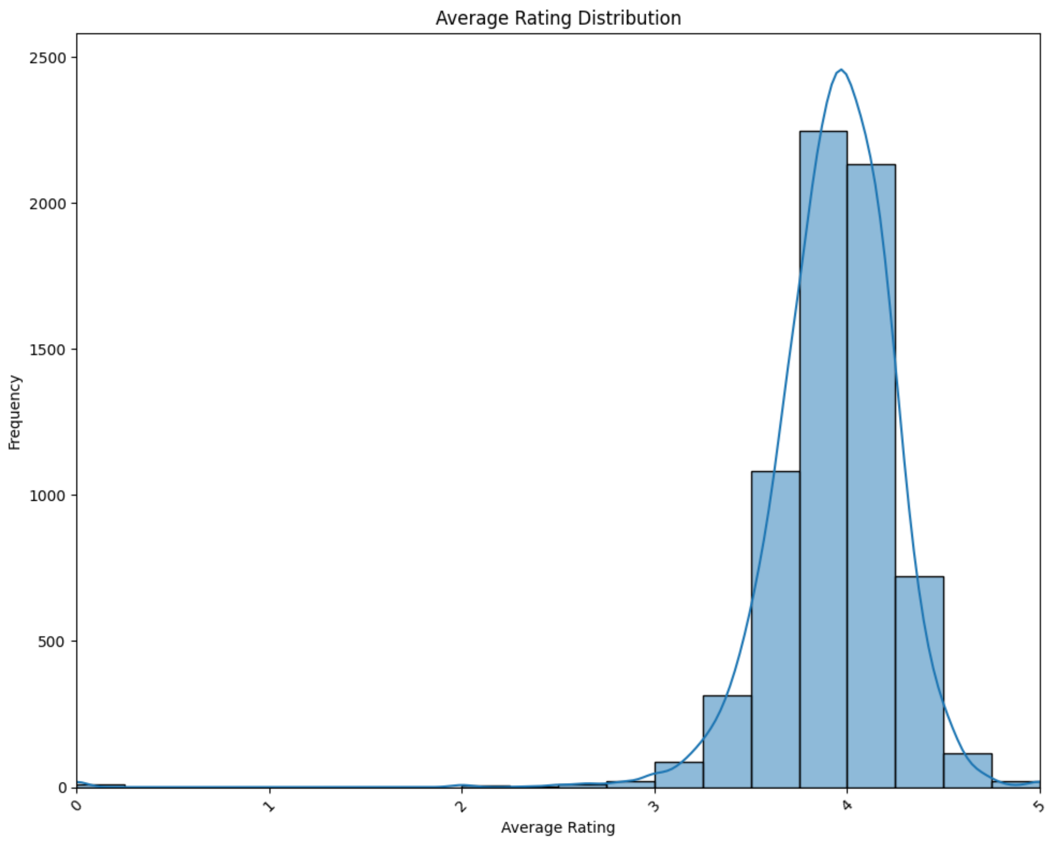


Figure C.4: Rating Distribution: Histogram of average book ratings. The skew toward higher ratings is typical for user-generated content and informed filter default settings.

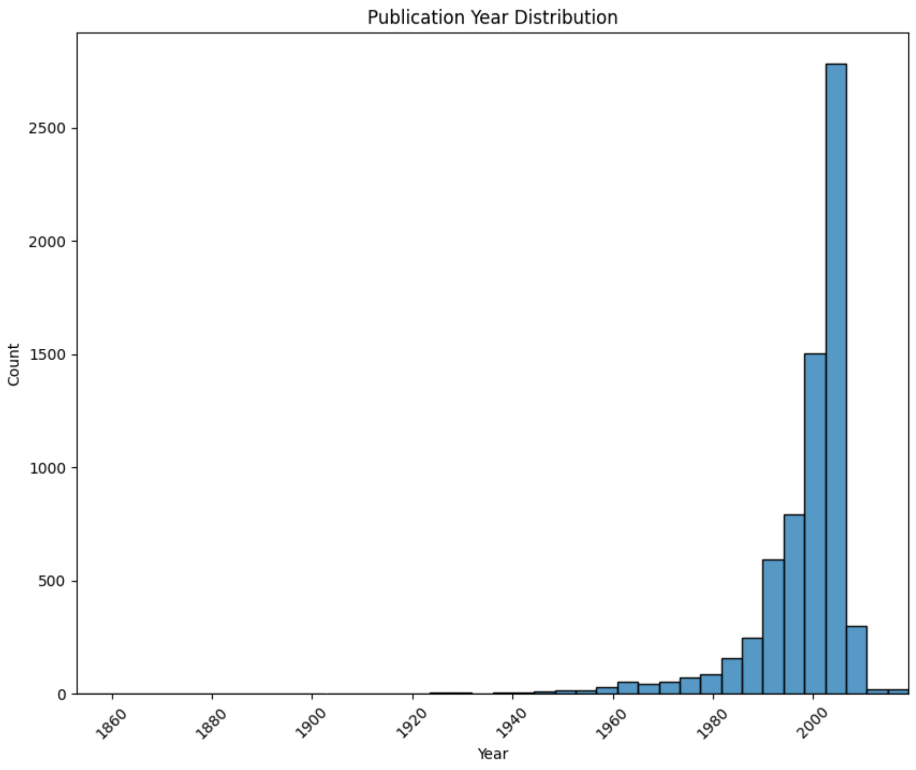


Figure C.5: Publication Year Distribution: Shows the number of books published per year. The majority are post-2000, indicating a recency bias that may impact topic coverage.

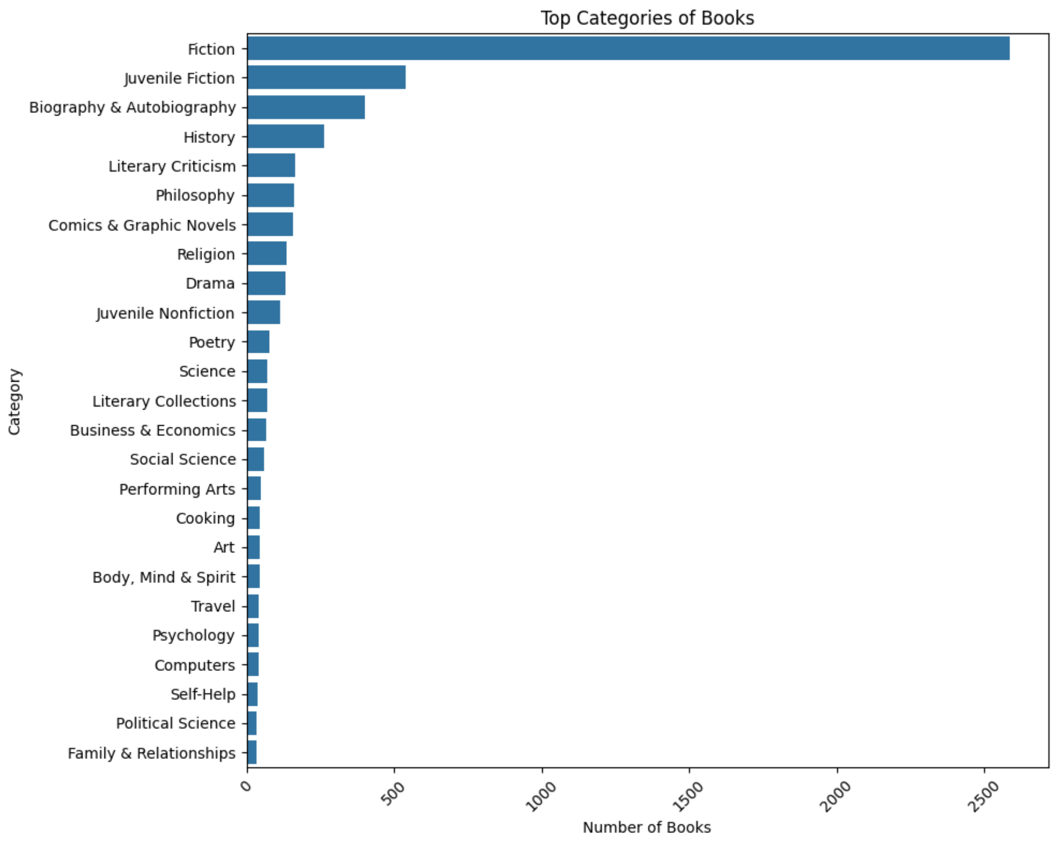


Figure C.6: Top Categories: A bar chart of the most frequent category labels in the original dataset. This informed the definition of candidate labels for zero-shot classification.

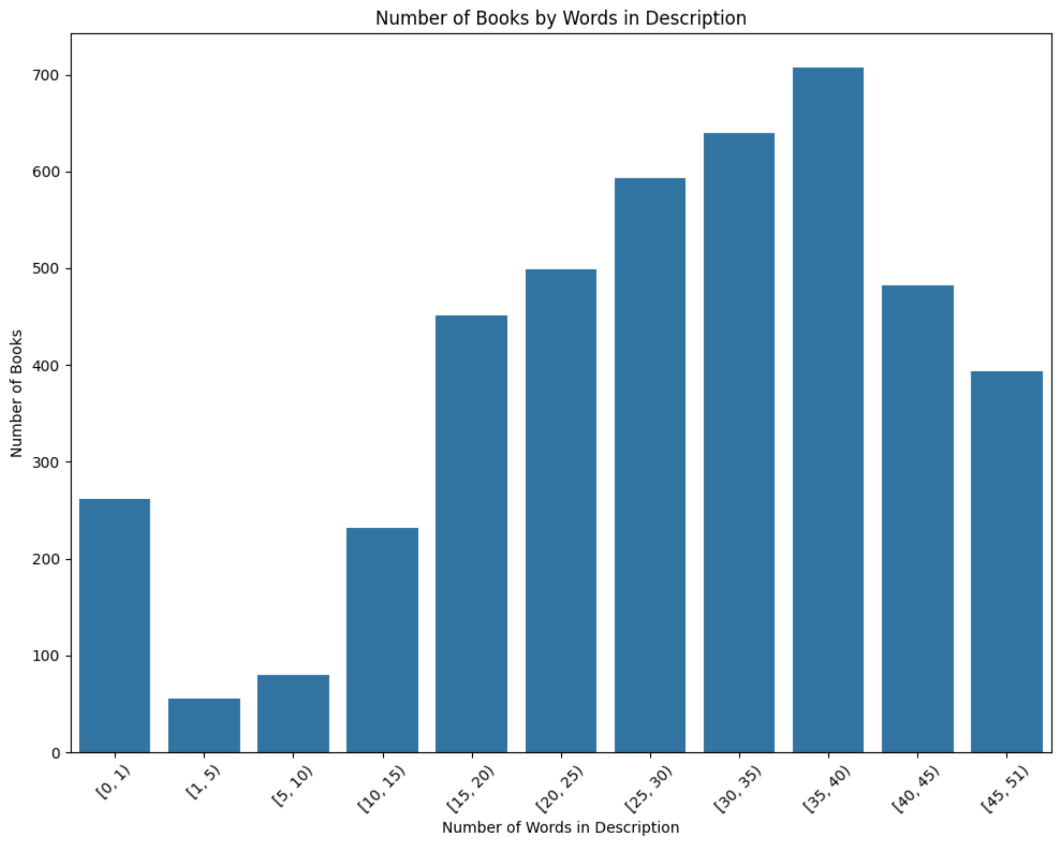


Figure C.7: Books with Short Descriptions: Distribution of books with fewer than 50 words in their descriptions. These entries were flagged for enrichment or removal.

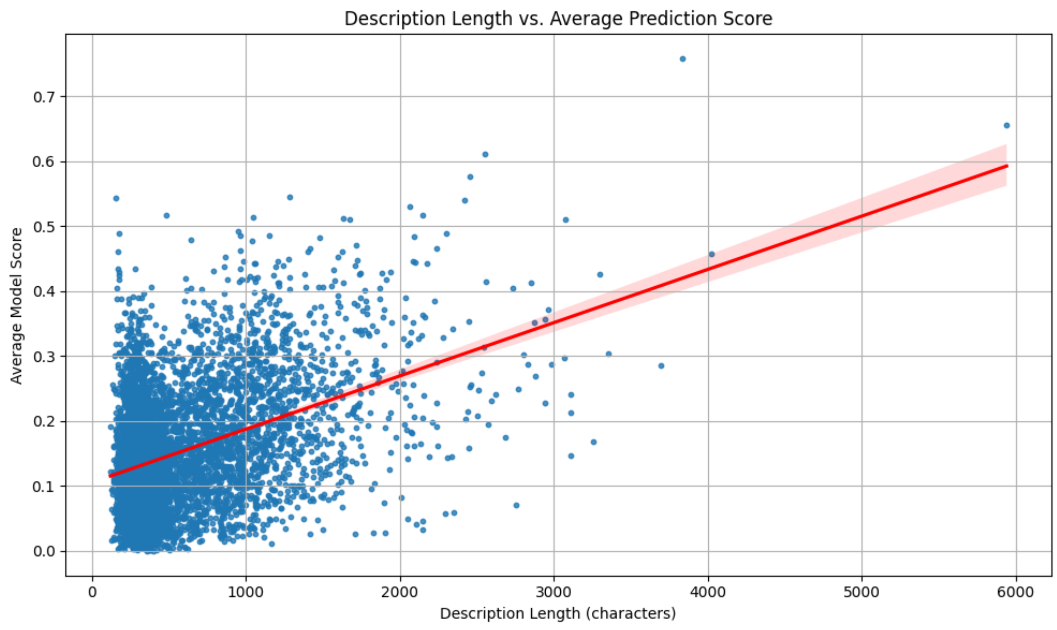


Figure C.8: Description Length vs. Confidence Score: A regression plot showing the positive correlation between description length and average model confidence. This supported the threshold of 200 characters.

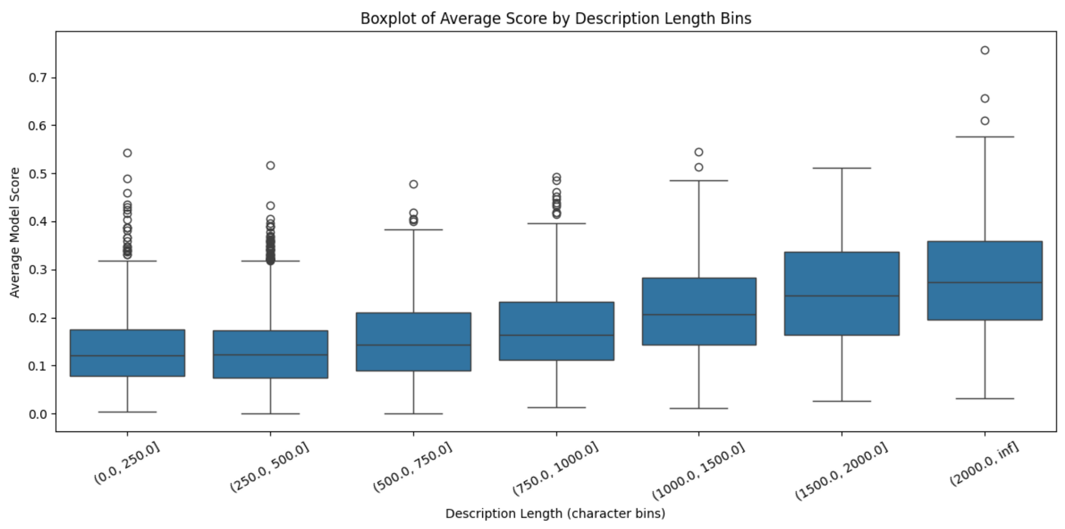


Figure C.9: Score by Length Bin: A boxplot displaying score variation across binned description lengths. Helps visualize consistency and reliability of model outputs by text richness.

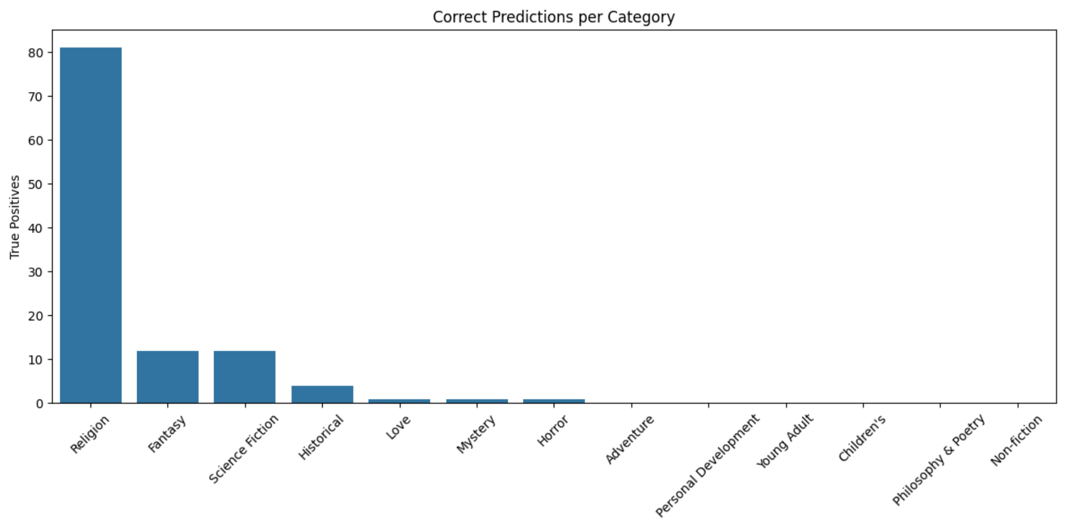


Figure C.10: Prediction Matches per Category: Bar chart of true positives per category. Reflects model strength in domains like Fantasy and Love and highlights where fallback logic is critical.

D
Theory of Sentence Embeddings

This appendix expands on the theoretical and practical foundations of sentence embeddings. These representations serve as the basis for semantic similarity throughout the system, as discussed in Chapter 5. Understanding how they are constructed and used is essential for grasping the logic behind embedding-based search and classification.

D.1. From Words to Vectors

In traditional NLP, words are first tokenized and represented numerically. Early methods like one-hot encoding fail to capture semantic relationships between words. This motivated the development of distributed representations — where words are embedded in continuous vector spaces.

Word embeddings like Word2Vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014) map words with similar contexts to nearby points in a high-dimensional space. However, they produce static embeddings: the word "bank" will have the same vector whether referring to a financial institution or a riverbank.

D.2. Sentence Embeddings

Sentence embeddings aim to extend this idea to whole sentences or paragraphs. They encode variable-length text into fixed-length dense vectors that capture meaning, context, and structure.

Formally, let:

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

where t is the input sentence, f is an embedding function (typically a neural model), and d is the dimensionality of the embedding space.

D.3. Transformer-Based Embeddings

Transformer models (Vaswani et al., 2017) have become the backbone of modern NLP. Instead of relying on fixed context windows, they use self-attention to consider the entire sentence when encoding each word.

Sentence-BERT (SBERT) (Reimers & Gurevych, 2019) fine-tunes transformer architectures to generate sentence-level embeddings. This project uses the distilled variant `all-MiniLM-L6-v2`, which balances semantic performance with CPU efficiency.

Architecture Overview

Below is a simplified diagram of the embedding process using a transformer:

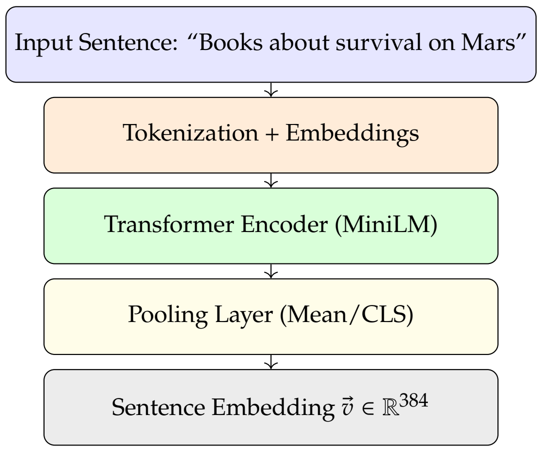

Input Sentence: "Books about survival on Mars"
Tokenization + Embeddings
Transformer Encoder (MiniLM)
Pooling Layer (Mean/CLS)
Sentence Embedding $\vec{v} \in \mathbb{R}^{384}$

Figure D.1: Pipeline for generating a sentence embedding

D.4. Similarity Measures

Once embedded, sentences can be compared using vector similarity. Common measures include:

- **Cosine similarity:**
$$\cos(\theta) = \frac{\vec{v}_1 \cdot \vec{v}_2}{\|\vec{v}_1\|\|\vec{v}_2\|}$$
Measures the angle between two vectors, emphasizing direction over magnitude.
- **Euclidean (L2) distance:**
$$\|\vec{v}_1 - \vec{v}_2\|_2^2$$
Measures straight-line distance. This project uses L2 distance for compatibility with FAISS.

D.5. Intuitive Example

Consider these two sentences:

> *"A man is walking a dog."*
> *"Someone strolls through the park with a pet."*

Despite having no identical words, their embeddings lie close together in semantic space, enabling meaningful comparisons beyond keyword overlap.

D.6. Limitations and Challenges

While powerful, sentence embeddings have limitations:

- Cannot perfectly disambiguate sarcasm, irony, or negation
- Performance degrades on domain-specific jargon
- Quality depends heavily on description clarity and length

D.7. Conclusion

Sentence embeddings are a foundational tool for semantic search and classification. By transforming language into geometry, they unlock flexible, content-aware systems that work without user histories or explicit labels.

Models like MiniLM make such capabilities available even on modest hardware — aligning with the project's offline-first design.

E
Vector Similarity and Search

This appendix elaborates on the mathematical foundations and implementation considerations behind vector similarity search — the mechanism used to match queries and books in semantic space.

E.1. Motivation

Traditional search engines rely on exact term overlap. However, semantically similar phrases often differ lexically:

> *"Books about exploring loneliness in space"* vs. *"Isolation in science fiction"*

Embedding-based systems overcome this limitation by projecting both queries and documents into a shared vector space. The task then becomes: *find the vectors nearest to a query vector*.

E.2. Mathematical Background

Let $\vec{q} \in \mathbb{R}^d$ represent a query embedding, and let $\{\vec{b}_1, \vec{b}_2, \dots, \vec{b}_n\} \subset \mathbb{R}^d$ be the embeddings of all books.

The most similar books are found by minimizing the distance function:

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

Alternatively, cosine similarity can be used:

$$\cos(\theta) = \frac{\vec{q} \cdot \vec{b}_i}{\|\vec{q}\|\|\vec{b}_i\|}$$

In this system, squared Euclidean (L2) distance is used to match the FAISS index type.

E.3. Geometric Intuition

In geometric terms, the embedding space is like a multi-dimensional landscape. Each book occupies a fixed location. A query becomes a "probe," and similarity search identifies the closest "peaks."

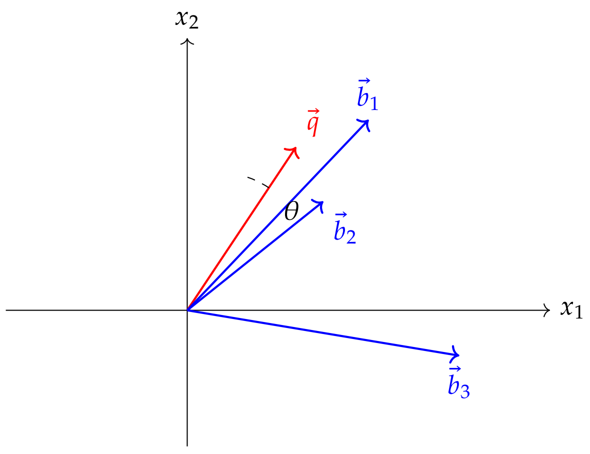


Figure E.1: Visualizing semantic proximity between query and book vectors

The smaller the angle or distance, the more semantically relevant the book is.

E.4. Indexing with FAISS

FAISS (Johnson et al., 2021) is an open-source library from Facebook AI that enables efficient similarity search over high-dimensional vectors.

This project uses: - `IndexFlatL2` — brute-force, exact L2 search - CPU-only backend for offline execution - Sub-second retrieval of top-k results from over 5,000 book vectors

For large datasets, approximate indices (e.g., HNSW, IVF) can be substituted for faster but slightly less precise searches.

E.5. Advantages of Vector Search

- **Language-aware:** Finds relevant matches even with low lexical overlap
- **Scalable:** Compatible with millions of documents with indexing
- **Offline-capable:** Entire process runs locally without cloud dependencies

E.6. Limitations

- **Semantic ambiguity:** Vector proximity does not guarantee relevance in all cases
- **Cold start:** Requires embeddings for all documents up front
- **Interpretability:** Search decisions are hard to explain due to vector abstraction

E.7. Conclusion

Vector similarity search replaces keyword matching with a geometric proximity task in semantic space. Combined with embedding models, this enables robust content-based recommendations even in resource-constrained, offline environments.

F
Theory of Zero-Shot Classification

This appendix elaborates on zero-shot classification (ZSC), which is used in this system to infer thematic categories for books. Unlike traditional classification methods, ZSC does not require labeled training data for each class. Instead, it relies on a language model's ability to understand and reason about class labels as natural language.

F.1. What is Zero-Shot Classification?

Traditional classification models require supervised training data — examples of each class to learn from. Zero-shot classification removes this constraint. A model trained on natural language inference (NLI) is reused for classification by rephrasing the task as a premise–hypothesis pair.

Premise: "This book is about elves and ancient magic."
Hypothesis: "This text is about fantasy."

The model estimates the probability that the hypothesis is entailed by the premise. If high, the label ("fantasy") is considered appropriate.

F.2. Model Foundations

This project uses `facebook/bart-large-mnli`, a transformer model fine-tuned on the MultiNLI corpus (Williams et al., 2018). The model performs classification by computing entailment scores between the input and each label reformulated as a natural sentence.

Architecture Overview

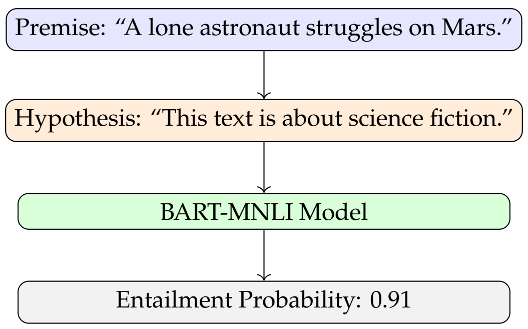

Premise: "A lone astronaut struggles on Mars."
Hypothesis: "This text is about science fiction."
BART-MNLI Model
Entailment Probability: 0.91

Figure F.1: Zero-shot classification via textual entailment

F.3. How It Works in Practice

Given a book description and a list of candidate genres (e.g., "fantasy", "science fiction", "romance"), each label is evaluated independently as a hypothesis. Labels with high entailment scores are retained.

This approach enables:

- Multi-label classification: multiple genres can apply
- Dynamic label sets: new genres can be introduced without retraining
- Minimal supervision: no need for annotated datasets

F.4. Confidence Thresholding

The model returns a confidence score for each label. Labels below a threshold (e.g., 0.4) are discarded to reduce noise. Labels just below the threshold may be added through fallback keyword rules.

F.5. Fallback Strategies

To improve coverage when ZSC fails, a rule-based system checks for genre-specific keywords in the description or title. This hybrid approach balances model generalization with deterministic recall.

F.6. Limitations

- **Surface sensitivity:** ZSC may be confused by subtle phrasing
- **High compute cost:** Each label requires a full model pass
- **Bias leakage:** Pretrained models may carry genre biases

F.7. Conclusion

Zero-shot classification enables genre inference without labeled examples, making it ideal for low-resource, flexible systems. By treating labels as hypotheses, it leverages the general reasoning ability of transformer models — aligning well with the offline and adaptable design goals of this project.

G
Data Cleaning and Augmentation

This appendix explores the techniques used for cleaning and augmenting book metadata, which was essential for building a semantically rich, searchable dataset. It addresses how inconsistencies, missing values, and weak textual descriptions were handled to support effective downstream ML tasks.

G.1. Why Data Quality Matters

In embedding-based systems, input quality directly affects output relevance. Ambiguous or sparse descriptions produce poor embeddings, reducing semantic precision. Similarly, inconsistent or missing metadata can weaken classification and filtering performance.

G.2. Cleaning Pipeline Overview

The dataset originally included over 7,000 book records, aggregated from various sources with variable completeness. The following steps were applied:

1. Dropped entries missing title or description
2. Removed duplicates and outliers (e.g., unusually long texts)
3. Normalized text casing and whitespace
4. Replaced null fields with placeholders or estimations
5. Removed boilerplate text (e.g., "No description available")

Cleaning Heuristics

These heuristics were used to streamline cleaning:

- Length thresholds (e.g., min 100 characters for description)
- Filtering based on punctuation density (flagging auto-generated entries)
- Removing descriptions composed entirely of keywords or tags

G.3. Data Augmentation

To improve the semantic richness of inputs, the following augmentation methods were used:

(1) Composite Description Field

A new field `search_text` was created by merging:

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

This gave more context to the embedding model, emphasizing title and author relevance.

(2) External Source Integration
Descriptions were extended where possible using public APIs:

- Open Library API
- Google Books API

If a book had a matching identifier (e.g., ISBN), missing fields like description, page count, or rating were filled in.

(3) Metadata Derivation
Additional fields were derived:

- **Book age:** computed from publication year
- **Page count bins:** converted to categorical buckets
- **Rating level:** numerical rating mapped to qualitative tiers

G.4. Correlation Checks

Correlation analysis between description length and classification confidence was conducted to justify filtering. Spearman's ρ and Pearson's r were computed and visualized to support design decisions (see Appendix C).

G.5. Fallback Design Philosophy

Where automatic tools failed (e.g., ZSC output too sparse), manual logic was introduced:

- Keyword-based genre tagging (e.g., "spaceship" $\rightarrow$ Sci-Fi)
- Heuristic label propagation within author clusters

This hybrid design emphasizes robustness over purity — especially valuable for offline and low-resource contexts.

G.6. Limitations and Trade-offs

- Augmentation may introduce minor inconsistencies (e.g., mismatched genres across sources)
- Some fields were inferred from partial matches and are best-effort estimates
- Reliance on third-party APIs, while limited, introduces variation in format and tone

G.7. Conclusion

Data preparation is foundational to the success of semantic systems. The cleaning and augmentation pipeline ensured that even lightweight models received rich, structured inputs — enabling effective classification and retrieval downstream.

H
Local-First System Design Principles

This appendix outlines the principles behind local-first architecture, as applied to this project's end-to-end recommendation system. It explains how privacy, autonomy, and offline usability influenced the choice of tools and methods.

H.1. What is Local-First Software?

Local-first software prioritizes performing all computation and storage on the user's own device. Network access is optional, not essential. Key motivations include:

- **Privacy:** No personal data is sent to remote servers.
- **Reliability:** The system functions without internet access.
- **Ownership:** The user retains full control over their data and execution.

This paradigm aligns well with AI applications where model inference is possible without cloud dependencies.

H.2. Application to This Project

Every step of the pipeline was designed to run locally:

- **Model inference:** All embedding and classification done using CPU-friendly models (MiniLM, BART-MNLI).
- **Storage:** All data and embeddings stored in local CSV and FAISS index files.
- **Interface:** The Streamlit app runs entirely within a browser on localhost.

H.3. Architectural Choices

The system was built around modular, loosely coupled components:

1. Python scripts for data ingestion, augmentation, and embedding
2. FAISS index for efficient vector search
3. Streamlit frontend for interaction

Each step can be re-run or updated independently, supporting rapid experimentation and debugging.

H.4. Trade-Offs and Limitations

- **No personalization:** Without user tracking or history, recommendations are purely content-based.
- **Compute constraints:** Model size is limited by local CPU/RAM availability.

- **No online learning:** The system does not evolve based on user behavior unless manually re-trained.

These limitations are acceptable within the project's scope and enable greater user autonomy.

H.5. Offline AI: Why It Matters

Running AI locally is increasingly viable due to efficient models and improved hardware. Benefits include:

- Resilience in low-connectivity environments (e.g., education, fieldwork)
- Cost reduction (no cloud inference or hosting fees)
- Alignment with GDPR and user-rights perspectives

H.6. Comparison with Cloud-Based Systems

Aspect	**Local-First**	**Cloud-Based**
Latency	Sub-second (no network delay)	Depends on API round-trips
Privacy	No data leaves device	Data often logged/transferred
Scalability	Limited to local hardware	Scales elastically with demand
Update Mechanism	Manual or scripted	Centralized and automatic
Dependency	Self-contained	Relies on service availability

H.7. Conclusion

Local-first design makes AI more accessible, ethical, and sustainable. By avoiding network reliance, the system empowers users to explore semantic search capabilities on their own terms, without compromising performance or flexibility.

I

Evaluation and Performance Metrics

This appendix describes the evaluation strategy used to assess the performance of the local book recommendation system. Since no ground-truth labels or user feedback data were available, traditional supervised metrics (e.g., accuracy, precision, recall) could not be applied. Instead, alternative metrics were used to evaluate semantic relevance, system responsiveness, and overall usability.

I.1. Constraints on Evaluation

- **No user history:** No personalized or collaborative filtering baseline.
- **No labeled test set:** No "correct" genre or match annotations.
- **Offline-only:** All testing conducted without server-side analytics or live monitoring.

This necessitated a human-in-the-loop, qualitative approach.

I.2. Evaluation Criteria

Four core criteria guided system evaluation:

1. Semantic Relevance

Assessed whether the top-k results for a query were semantically aligned with its intent. This was evaluated manually using test queries such as:

> *"Books about surviving in space"* $\rightarrow$ Check that returned books involve space travel, isolation, or survival themes.

In most test cases, at least 4 of the top 6 results were contextually appropriate.

2. Filtering Utility

Tested whether genre and rating filters improved or restricted result precision. Filters were toggled and queries rerun to compare:

- Breadth of result diversity
- Preservation of relevance post-filter

This highlighted that over-filtering may remove semantically correct results.

3. Responsiveness

Measured latency of key operations on CPU-only hardware:

- Query embedding: 150ms

- FAISS search (top 60): <50ms
- Filtering and rendering: <300ms

Sub-second feedback ensured smooth interactivity.

4. Offline Functionality

Validated that all components (embedding, classification, filtering, UI) function without network access. This included running:

- Zero-shot classification with cached local models
- Full semantic search using only local vector index
- No calls to online services or third-party APIs

I.3. Example Query Types

Different categories of queries were tested:

- **Thematic:** "Books about AI ethics"
- **Mood-driven:** "Something dark and mysterious"
- **Specific:** "Books like Dune"

Semantic embedding enabled flexible and expressive querying even when keywords were absent.

I.4. Limitations of Evaluation

- No user feedback loop to refine metrics or iterate design
- No A/B testing against alternatives (e.g., TF-IDF or keyword search)
- Evaluation is largely anecdotal and lacks quantitative validation

Nevertheless, the qualitative results were consistent and satisfactory for the project scope.

I.5. Conclusion

Evaluation in low-data, offline-first ML projects requires adapted criteria. By focusing on semantic quality, system speed, and functional constraints, meaningful insights were still obtained. The framework can be extended in future work with automated tests or user trials.