

Final Report on Advanced Book Search & Recommender APIs

May 15, 2024

Abstract

This report presents a detailed development summary of the Advanced Book Search and Recommendations system. The system was created with the aim of providing fast, relevant, and personalized recommendations and search algorithms to all manner of bookstores, libraries, and other facilities. This report details the research, development, testing, and deployment stages of the project.

Acknowledgement

We owe great thanks to many people who helped and supported us during the designing and completing our project as well as the writing of this report also. First and foremost, the gratitude goes to our project supervisor Dr. S. Kirushanth, who encouraged, guided, and supported us to complete each milestone of our project from the beginning to the end. And also, we thank our friends, who have shared their knowledge and experiences for our project development. Apart from the efforts of the project team, the success of the project depends on the encouragement and guidelines given by many others. We also thank our seniors, colleagues, and instructors of the Faculty of Applied Science. Finally, we thank everyone who guided us towards the success.

Thank you.

Declaration

We hereby declare that this report submitted for evaluation of the course module IT3162 leading to the award of Bachelor of Information Technology is entirely our own work and the contents taken from the work of others has been cited and acknowledged within the text. This report has not been submitted for any degree in this university or any other institution.

Reg.No	Name	Signature
2019/ICT/24	Mithini Rathnayake	
2019/ICT/35	M.A.Ihsana	
2019/ICT/46	S.G.Seyone	
2019/ICT/64	Charitha Dilman	
2019/ICT/66	Tharushi Nimnadi	
2019/ICT/85	Ashma Sandeepa	
2019/ICT/98	Fathima Zeena	
2019/ICT/108	Sonali Kalpani	

I hereby certified that I have supervised this project.

Dr.S.Kirushanth
Supervisor
Senior Lecturer
Head of the department
Department of Physical Science,
Faculty of Applied Sciences.

Date

Contents

1	Introduction	5
2	Problem statement	5
3	Literature Review	6
4	Methodology	6
4.1	Development	12
4.1.1	API Creation	12
4.1.2	Demo Application Creation	17
4.1.3	Bookstore Application creation	18
4.2	Testing	20
4.3	Deployment	31
4.4	Deploying the API Server	31
4.5	Deploying the Demo Application	32
5	Discussion	32
6	Conclusion	36
	References	37

1 Introduction

The digital age has revolutionized the way we interact with literature, providing unprecedented access to a vast array of books through online platforms. However, despite the convenience of digital bookstores, users often encounter challenges in navigating the plethora of available titles, finding books that match their interests, and ensuring the authenticity of transactions, particularly in the realm of used book sales. Addressing these challenges, our project focuses on the development of a used book sales system powered by blockchain technology. By integrating blockchain, we aim to enhance transparency, security, and trust in the used book marketplace, mitigating concerns related to counterfeit books and fraudulent transactions. Furthermore, by incorporating content-based search algorithms, our system offers personalized recommendations to users, guiding them towards books that align with their tastes and preferences. This report provides a comprehensive overview of our project, detailing the objectives, methodology, design, development, testing, deployment, and future prospects of our innovative solution. Through this endeavor, we seek to revolutionize the used book marketplace, fostering a seamless and trustworthy experience for book enthusiasts worldwide.

2 Problem statement

The domain of online bookstores, encompassing both new and used books, constitutes a significant segment of the e-commerce industry. However, the current landscape of online book sales platforms, particularly in the realm of used books, presents several challenges that hinder user experience and trustworthiness. Existing solutions often lack robust mechanisms for facilitating efficient search and discovery processes, resulting in user frustration and suboptimal outcomes. Users encounter difficulties in navigating through the vast array of available titles, and concerns regarding the authenticity and provenance of books persist, casting doubt on the reliability of transactions.

Inadequate search algorithms exacerbate the problem, failing to provide personalized recommendations tailored to individual user preferences. As a result, users struggle to find relevant titles amidst the extensive catalog, hampering their overall experience. Furthermore, the management of bookstores faces challenges in efficiently handling inventory and ensuring the quality and authenticity of used books.

To address these challenges, our project proposes a comprehensive solution that leverages blockchain technology and advanced search algorithms. By integrating blockchain, we aim to enhance transparency, security, and trust in the used book marketplace. Blockchain ensures immutable records of transactions, mitigating concerns related to counterfeit books and fraudulent activities. Additionally, our solution focuses on improving the performance of search algorithms to provide users with personalized recommendations, facilitating smoother transactions and enhancing user satisfaction.

In summary, our project seeks to revolutionize the way users interact with online bookstores by addressing the inefficiencies and challenges prevalent in the current landscape. By providing a robust and user-friendly platform for buying and selling used books, we aim to foster a seamless and trustworthy experience for both book enthusiasts and bookstore management. Used references are BookSwap.lk and UsedBooks.lk

3 Literature Review

4 Methodology

Prior to proposal presentation, we created system block diagrams for how the various systems are to function within the system.

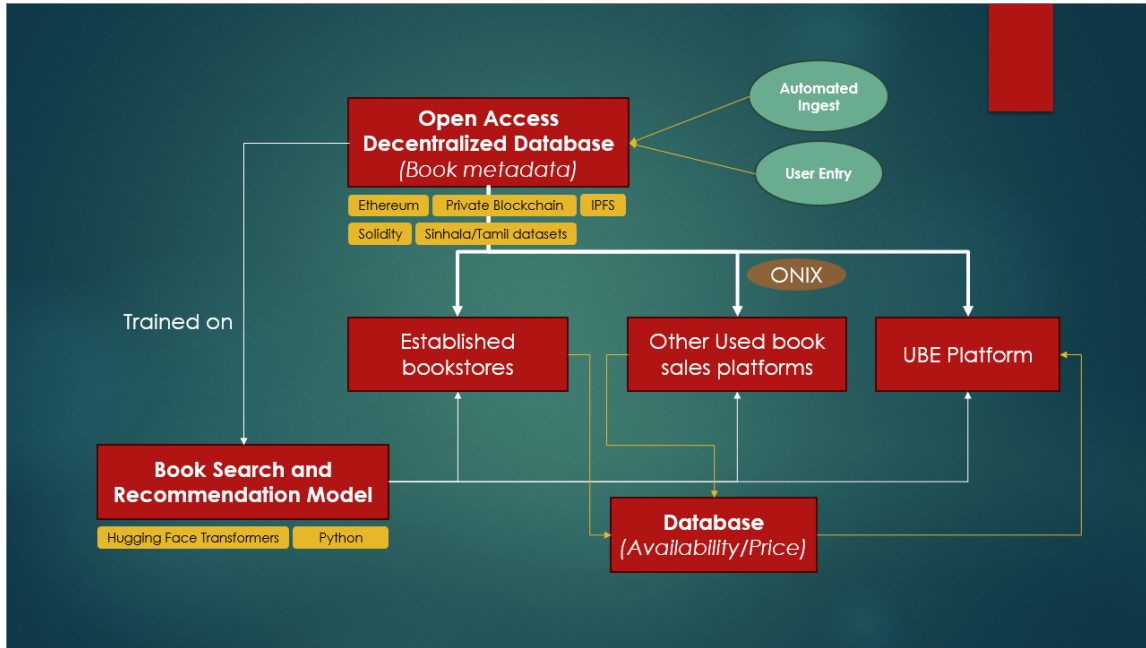


Figure 1: Block Diagram for Proposed System

As per our proposal, we identified 4 key deliverables necessary to succeed in our project, namely:

- Book Data Database
- Recommendation System APIs
- Search APIs
- Demo applications

Initially we began our project with data collection, analysis, and planning. We obtained book metadata from various sources, chiefly,

- Libgen.rs database dumps
- malcolmosh's excellend goodbooks-10k-extended dataset
- Sri Lanka National Library's Sri Lanka National Bibliography

The first two sources were preformatted and easy to use data, which was used to train our base recommender model, which functions in English language. We were unable to fully exploit the SLNB data, as data extraction proved to be beyond our current capacity.

Utilizing the pdfplumber package in Python, we were able to extract tokens for every word in the bibliography. Here's a sample of the extracted text.

English Text: ['SRI', 'LANKA', 'NATIONAL', 'BIBLIOGRAPHY', 'SUBJECT', 'SECTION', 'COMPUTER', 'SCIENCE', 'SOCIAL', 'SCIENCES', 'INFORMATION', 'GENERAL', 'WORKS', 'Communication', 'Mysteries', 'Critical', 'perspectives', 'on', 'open', 'development', 'empirical', 'interrogation', 'of', 'theory', 'Razeen', 'Muhammad', 'construction', 'ed', 'Arul', 'Chib', 'Caitlin', 'M', 'The', 'wonders', 'of', 'the', 'world', 'Muhammad', 'Bentley', 'and', 'Matthew', 'L', 'Smith', 'England', 'Razeen', 'tr', 'by', 'A', 'C', 'M', 'Wabanbey', 'The', 'MIT', 'Press', 'xi', 'p', 'cm', 'Kurunegala', 'Author', 'p', 'The', 'MIT', 'press', 'international', 'development', 'photos', 'cm', 'research', 'centre', 'series', 'Ad', 'Bc', 'Rs', 'NL', 'Ad', 'Bc', 'Unpriced', 'NL', 'ISBN', 'ISBN', 'Curiosities', 'and', 'wonders', 'Economic', 'development', 'Information', 'commons', 'Information', 'society', 'Information', 'technology', 'Social', 'aspects', 'RELIGION', 'Open', 'source', 'software', 'Sri', 'Lanka', 'Standard', 'Institution', 'OTHER', 'RELIGIONS', 'Sri', 'Lanka', 'Sri', 'Lanka', 'Somasundara', 'J', 'W', 'D', 'Up', 'country', 'Tamils', 'charting', 'a', 'new', 'future', 'in', 'The', 'most', 'sacred', 'Sri', 'Pada', 'J', 'W', 'D', 'Sri', 'Lanka', 'ed', 'Daniel', 'Bass', 'and', 'B', 'Somasundara', 'and', 'H', 'M', 'Jayantha', 'Skanthakumar', 'Colombo', 'International', 'Wijeratna', 'tr', 'by', 'Sunil', 'Wijeyesinghe', 'Centre', 'for', 'Ethnic', 'Studies', 'Colombo', 'S', 'Godage', 'p', 'xxi', 'p', 'cm', 'photos', 'cm', 'Ts', 'Cbc', 'Unpriced', 'NL', 'Ad', 'Bc', 'Rs', 'NL', 'ISBN', 'ISBN', 'NA', 'Malaiyaha', 'Tamil', 'Sri', 'Lankan', 'people', 'Adam', 's', 'Peak', 'Sri', 'Lanka', 'Sri', 'Lanka', 'Mountains', 'Sri', 'Lanka', 'Race', 'relations', 'Political', 'aspects', 'Sri', 'Lanka', 'Ethnic', 'relations', 'Sri', 'Lanka', 'Politics', 'and', 'government', 'POLITICAL', 'SCIENCE', 'Sri', 'Lanka', 'Perera', 'L', 'C', 'Facing', 'our', 'own', 'citizens', 'in', 'conflict', 'situations', 'the', 'Southern', 'JVP', 'insurrection', 'L', 'C', 'Perera', 's', 'l', 's', 'n', 'p', 'cm', 'Ad', 'Bc', 'Rs', 'NL', 'ISBN', 'NA', 'Janatha', 'Vimukthi', 'Peramuna', 'Sri', 'Lanka', 'Politics', 'and', 'government', 'ECONOMICS', 'PUBLIC', 'ADMINISTRATION', 'AND', 'MILITARY', 'SCIENCE', 'Organizations', 'Annual', 'reports', 'Sri', 'Lanka', 'Piramal', 'Glass', 'Ceylon', 'PLC', 'Annual', 'report', 'Ratmalana', 'Gamaathige', 'Amarasena', 'Piramal', 'Glass', 'Ceylon', 'PLC', 'Making', 'project', 'administration', 'efficient', 'and', 'p', 'cm', 'effective', 'challenges', 'and', 'good', 'practices', 'Ad', 'Bc', 'Unpriced', 'NL', 'Amarasena', 'Gamaathige', 's', 'l', 'Author', 'NA', 'xxii', 'p', 'cm', 'Ad', 'Bc', 'Rs', 'NL', 'Business', 'enterprises', 'Sri', 'Lanka', 'ISBN', 'Periodicals', 'Economic', 'assistance', 'Sri', 'Lanka', 'Economic', 'development', 'projects', 'Sri', 'Lanka', 'Sri', 'Lanka', 'International', 'economic', 'relations', 'Made', 'in', 'Sri', 'Lanka', 'exporters', 'showcase', 'Colombo', 'Media', 'Services', 'Private', 'Limited', 'p', 'photos', 'x', 'cm', 'Ts', 'Bc', 'Rs', 'NL', 'TECHNOLOGY', 'ISBN', 'MEDICINE', 'AND', 'HEALTH', 'Economic', 'development', 'Pictorial', 'works', 'Review', 'and', 'exercise', 'Sri', 'Lanka', 'Economic', 'conditions', 'Pictorial', 'works', 'Fonseka', 'Palitha', 'Model', 'questions', 'for', 'pharmacists', 'examinations', 'part', 'multiple', 'choice', 'LAW', 'questions', 'Palitha', 'Fonseka', 'Gampaha', 'NPI', 'Publications', 'p', 'cm', 'Lawyers', 'Ad', 'Bc', 'Rs', 'NL', 'ISBN', 'Pursuing', 'a', 'vision', 'of', 'justice', 'essays', 'in', 'honour', 'of', 'Maithri', 'Panagoda', 'ed', 'Senaka', 'Pharmacology', 'Problem', 'exercise', 'etc', 'Weeraratna', 'Colombo', 'Vijitha', 'Yapa', 'Publications', 'xxvii', 'p', 'photos', 'cm', 'Prescription', 'writing', 'Ad', 'Bc', 'Rs', 'NL', 'ISBN', 'Fonseka', 'Palitha', 'Pharmaceutical', 'calculations', 'Palitha', 'Lawyers', 'Sri', 'Lanka', 'Biography', 'Fonseka', 'Gampaha', 'NPI', 'Publications', 'Panagoda', 'Maithri',

'p', 'ill', 'cm', 'Ad', 'Bc', 'Rs', 'NL', 'ISBN', 'Pharmaceutical', 'chemistry', 'Pharmacology', 'Ayurvedic', 'medicine', 'AGRICULTURE', 'AND', 'RELATED', 'TECHNOLOGIES', 'Weerasekera', 'K', 'R', 'Review', 'of', 'Prameha', 'roga', 'in', 'ayurveda', 'Sri', 'Lanka', 'perspective', 'K', 'R', 'Weerasekera', 'Colombo', 'S', 'Godage', 'Myths', 'truths', 'of', 'oil', 'palm', 'a', 'study', 'on', 'p', 'cm', 'Impacts', 'and', 'opportunities', 'of', 'sustainable', 'Ad', 'Bc', 'Rs', 'NL', 'oil', 'palm', 'cultivation', 'in', 'Asia', 'Hong', 'Kong', 'ISBN', 'Solidaridad', 'Network', 'Asia', 'Limited', 'p', 'ill', 'cm', 'Diabetes', 'Ayurvedic', 'treatment', 'Ad', 'Bc', 'Unpriced', 'NL', 'Medicine', 'Ayurvedic', 'Oil', 'palm', 'Sri', 'Lanka', 'Palm', 'oil', 'industry', 'Sri', 'Lanka', 'Medical', 'virology', 'Palm', 'oil', 'Sri', 'Lanka', 'Gunasekera', 'Kamani', 'Basics', 'in', 'virology', 'Kamani', 'Gunasekera', 'THE', 'ARTS', 'Nugegoda', 'Sarasavi', 'Publishers', 'p', 'cm', 'ARCHITECTURE', 'Ad', 'Bc', 'Rs', 'NL', 'ISBN', 'NA', 'Ambience', 'contemporary', 'design', 'January', 'Colombo', 'Media', 'Services', 'Medical', 'virology', 'Private', 'Limited', 'p', 'ill', 'Virology', 'x', 'cm', 'Virus', 'diseases', 'Diagnosis', 'Ts', 'Bc', 'Rs', 'NL', 'Viruses', 'Classification', 'ISBN', 'Architecture', 'Sri', 'Lanka', 'Review', 'and', 'exercise', 'Architecture', 'Sri', 'Lanka', 'Pictorial', 'works', 'Dasanayake', 'Lanka', 'Aid', 'to', 'mcq', 'True', 'False', 'in', 'obstetrics', 'RECREATIONAL', 'AND', 'gynaecology', 'for', 'medical', 'undergraduates', 'PERFORMING', 'ARTS', 'Lanka', 'Dasanayake', 'Nugegoda', 'Sarasavi', 'Publishers', 'p', 'cm', 'Puzzles', 'and', 'puzzle', 'games', 'Ad', 'Bc', 'Rs', 'NL', 'ISBN', 'NA', 'Samarasinghe', 'A', 'Sam', 'Fun', 'with', 'puzzles', 'A', 'Sam', 'Samarasinghe', 'Gynnaecology', 'Problems', 'exercises', 'etc', 'Nugegoda', 'Sarasavi', 'Publishers', 'Obstetrics', 'Problems', 'exercises', 'etc', 'p', 'ill', 'cm', 'Ad', 'Bc', 'Rs', 'NL', 'ISBN', 'NA', 'Intelligence', 'tests', 'Puzzles', 'Samarasinghe', 'A', 'Sam', 'I', 'like', 'puzzels', 'A', 'Sam', 'Samarasinghe', 'Nugegoda', 'Sarasavi', 'Publishers', 'p', 'ill', 'cm', 'Ad', 'Bc', 'Rs', 'NL', 'ISBN', 'NA', 'Intelligence', 'tests', 'Puzzles', 'Samarasinghe', 'A', 'Sam', 'Wickramasekara', 'Eshani', 'IQ', 'puzzles', 'A', 'Sam', 'Samarasinghe', 'Dear', 'hearts', 'a', 'collection', 'of', 'poems', 'for', 'life', 'Nugegoda', 'Sarasavi', 'Publishers', 'Eshani', 'Wickramasekara', 's', 'l', 'Author', 'p', 'ill', 'cm', 'p', 'cm', 'Ad', 'Bc', 'Rs', 'NL', 'Ad', 'Bc', 'Unpriced', 'NL', 'ISBN', 'NA', 'ISBN', 'Intelligence', 'tests', 'English', 'poetry', 'Puzzles', 'Fiction', 'Novel', 'Samarasinghe', 'A', 'Sam', 'Jayawardana', 'Tarindi', 'Tricky', 'puzzles', 'A', 'Sam', 'Samarasinghe', 'The', 'secret', 'club', 'Stella', 'and', 'the', 'vampire', 'Nugegoda', 'Sarasavi', 'Publishers', 'teachers', 'Tarindi', 'Jayawardana', 's', 'l', 'p', 'cm', 'Author', 'p', 'ill', 'cm', 'Ad', 'Bc', 'Rs', 'NL', 'Ad', 'Bc', 'Rs', 'NL', 'ISBN', 'NA', 'ISBN', 'NA', 'Intelligence', 'tests', 'English', 'fiction', 'Puzzles', 'Fiction', 'Short', 'stories', 'LITERATURE', 'Subramaniam', 'Alagu', 'Closing', 'time', 'with', 'added', 'personal', 'notes', 'ENGLISH', 'LITERATURE', 'Alagu', 'Subramaniam', 'ed', 'by', 'Premila', 'Thurairatnam', 'nd', 'ed', 'Jaffna', 'Commercial', 'Poetry', 'Press', 'Stores', 'p', 'cm', 'Gunawardane', 'Ruwan', 'First', 'published', 'in', 'Clouds', 'cliffs', 'a', 'collection', 'of', 'Haiku', 'Ad', 'Bc', 'Unpriced', 'NL', 'poems', 'in', 'English', 'Ruwan', 'Gunawardane', 'ISBN', 'Galle', 'Author', 'vii', 'p', 'ill', 'x', 'cm', 'Short', 'stories', 'English', 'Ts', 'Cbc', 'Rs', 'NL', 'ISBN', 'Children', 's', 'fiction', 'English', 'poetry', 'Perera', 'Dinasha', 'Lisabeth', 'The', 'great', 'adventure', 'the', 'magic', 'mirror', 'Gunawardane', 'Ruwan', 'Dinasha', 'Lisabeth', 'Perera', 'Colombo', 'Vijitha', 'No', 'pun', 'intended', 'a', 'collection', 'of', 'poems', 'Yapa', 'Publications', 'p', 'col', 'ill', 'Ruwan', 'Gunawardane', 'Galle', 'Author',

'cm', 'vii', 'p', 'cm', 'Ad', 'Bc', 'Rs', 'NL', 'Ws', 'Cbc', 'Rs', 'NL', 'ISBN', 'NA', 'ISBN', 'Children', 's', 'stories', 'English', 'English', 'poetry', 'LITERATURES', 'OF', 'OTHER', 'HISTORY', 'GEOGRAPHY', 'AND', 'SPECIFIC', 'LANGUAGES', 'AND', 'AUXILIARY', 'DISCIPLINES', 'LANGUAGE', 'FAMILIES', 'Sri', 'Lanka', 'Fiction', 'Novel', 'Ambience', 'contemporary', 'design', 'December', 'Silva', 'W', 'A', 'Colombo', 'Media', 'Service', 'Private', 'Vijayaba', 'Kollaya', 'coup', 'in', 'Kotte', 'Kingdom', 'Limited', 'p', 'photo', 'x', 'cm', 'of', 'Sri', 'Lanka', 'W', 'A', 'Silva', 'tr', 'by', 'Janaka', 'Ts', 'Bc', 'Rs', 'NL', 'Perera', 'Nugegoda', 'Sarasavi', 'Publishers', 'ISBN', 'p', 'cm', 'Ad', 'Bc', 'Rs', 'NL', 'Hotels', 'Sri', 'Lanka', 'Periodical', 'works', 'ISBN', 'NA', 'Historical', 'fiction', 'Sinhalese', 'Sinhalese', 'fiction', 'Sinhalese', 'fiction', 'Translations', 'into', 'English']

Utilizing the fitz package, we were able to extract a more structured representation of the data. The issues faced when analyzing this data will be explained in detail in the discussion section.

```

return None

# Specify the PDF file path and the page number you want to extract text from
pdf_path = 'slnb/2020-2029/SLNB 2022-01-12/SLNB-2022-06 V.60 June.pdf'
page_number = 39 # Change this to the desired page number

# Extract text from the specified page
text_from_page = extract_text_from_page(pdf_path, page_number)

# Print the extracted text
if text_from_page:
    print(text_from_page)
else:
    print("Failed to extract text from the specified page.")

```

Executed at 2024.05.15 11:46:58 in 203ms

--

891.483 - சிறுவர சிறுகதைகள்
அகபதீர, கித்சிறி
யதாழ்கதவி மைற்றும் ருஹஹுணு குமைதாரி /
கித்சிறி அகபதீர ; கமைதாழ. கபத்மதயதா
நதாக்கஸ சரஸவதி. - ககதாமுழ்பு :
எஸ. ககதாடகக சககதாதரதாரர்கள பிமரகவட்
லிமிட்டட், 2021. - ப. 56 ; ச.ம. 22.
ஒழு-மைஉ : ர. 300.00 (1095881 NL)
ISBN 978-624-00-0944-7 (471286 NA)
1. சிறுகமதகள ,சிங்களம் - தமிழ்

கமைதாழகபயர்ப்பு
894.811 - தேமிழ இலக்கியம்
894.8111 - கவிகதேகள்
அப்துல் குத்தாஸ, அக்கமரயூர்
மைற்றுமைதாரு மைதாபியதா / அக்கமரயூர்
அப்துல் குத்தாஸ. - அக்கமரப்பற்று :
ஆசிரியர், 2021. - ப. xii, 108 ; ச.ம. 21.
ஒழு-மைஉ : ர. 560.00
(1093457 NL)
ISBN 978-955-96942-7-4
1. கவிமதகள - தமிழ்
கபலிஸரன, கச்சு

11
Figure 2: Data extracted using fitz

Next, we began architecting the applications necessary to implement the system. We decided to use Git for version control. Though there were other options such as mercurial available, the widespread use of Git and familiarity with it prompted us to choose it as our VCS software of choice.

4.1 Development

Development was split into three phases. Namely,

- API Creation
- Demo Application Creation
- Testing & Integration

4.1.1 API Creation

We chose to run our API on Flask, for its performance and ease of use. Since we simply needed to deploy the model and have a few routes to describe the functionalities of the API, and to serve the models, we felt that a micro-framework like Flask would be the best.

The API contains the following routes.

```
/predict  
  
/models
```

`/predict` is a HTTP POST endpoint, and is used to serve the Recommender models themselves. This accepts two parameters, `title`, and `model`. `Title` is the name of the book for which we want recommendations. `Model` gives shorthand for the various models able to be served by this API. For now, we can serve `distilbert`, `distilbert_v2`, `bert`, `tf_idf`, and `word2vec` models.

`/models` is a HTTP GET endpoint, and simply returns a list of available models that can be used by the client to request recommendations.

Recommender Models

We tried various approaches in creating the recommender models. First, we tried traditional ML techniques to create a model that can recommend books based on similarities in content to other books. Our first model, `tf_idf`, ended up being our most stable and useful model.

During development of the models, we faced various setbacks. One pitfall we encountered was due to us using Google Colaboratory to test our models. As we exported the model, and imported the code to our API program, the dumped model (using `joblib`) would not run. We later figured out that the dumping has to be done within the same package as will be used in execution, and that the dumping has to be done from a separate python file (we created a file called `dump_model`), else it will not create the `__main__` function properly.

TF-IDF

`tf_idf` utilizes Term Frequency - Inverse Document Frequency to generate vector embeddings for the various fields used to compare books. Term Frequency-Inverse Document Frequency (TF-IDF) is a numerical statistic used in information retrieval and text mining to reflect the importance of a term in a document relative to a collection of documents (corpus). It is commonly used as a weighting factor in various text analysis tasks such as document classification, information retrieval, and text similarity calculation. In this case, we utilized TF-IDF to generate embeddings for the book titles and descriptions. This model also processes language codes, genres, and author data, but utilizes different methodologies to convert this data into embeddings. For genres, we use Multi-Label Binarization, and a hashing algorithm to compute embeddings for the author field. The libraries implementing these algorithms were taken from `sklearn`.

For this model, we utilized Cosine Similarity as our measure of similarity, as it provided the best recommendations as per (limited) user testing. Given below is the feature extraction portion of the TF-IDF Model.

Listing 1: Feature Extraction using traditional ML

```
class FeatureExtractor:
def extract_features(self, books_df_processed):
    vectorizer = TfidfVectorizer()

    # Count of unique authors
    count_of_unique_authors = books_df_processed['author'].nunique()

    hasher = FeatureHasher(n_features=count_of_unique_authors,
                           input_type='string')
    mlb = MultiLabelBinarizer()

    # Hash the authors
    # author_features = hasher.transform(books_df_subset['author'])

    # Binarize the genres column
    binarized_genres = mlb.fit_transform(books_df_processed['genres'])

    # One-hot encode the language_code
    books_df_subset = pd.get_dummies(books_df_processed, columns=['language_code'])

    # Vectorize the title column
    title_features = vectorizer.fit_transform(books_df_subset['title'])

    # Vectorize the description column
    description_features = vectorizer.fit_transform(books_df_subset['description'])
```

```

# Composite feature Vector
composite_feature_vector = hstack([binarized_genres ,
                                   title_features , description_features])

return composite_feature_vector

```

* Note that the Author embeddings are suppressed in this code, and it is suppressed from the model because the data in the original dataset is not clean enough, which resulted in improper recommendations when using that column.

Fuzzy TF-IDF

This model is a simple modification of the TF-IDF model, which adds a fuzzy logic layer to the algorithm. This ensures that the closest match to a given book title is found using fuzzy logic, and that book title is processed in the recommender algorithm. The reason we decided to go this way is because in the event the user makes a typo, or does not know the name of the book in the exact format that is in the database (for instance, **The Smell of Death: Death #3 (2003)**, the user might just type **The Smell of Death** or **The Seell of Dearth**)

Bert & DistilBert

BERT is a language model based on the transformer architecture, released in 2018 by researchers at Google. We intended to use BERT for generating embeddings for the data, as transformer based pre-trained models provide significant advantages over traditional ML methodologies for extracting richer relationships between text tokens.

DistilBert is a newer model released by Google that is 40% smaller than Bert, yet provides 95% of its functionality. We were able to get a rudimentary model based on DistilBert running once, but without all the parameters it was ultimately useless.

While this was the idea in theory, we were unable to actually train any of the BERT based models, as the hardware available to us was not sufficient to train the model in any reasonable length of time. The best machine available at our disposal was a laptop equipped with an AMD Ryzen 9 6900HX processor, 16 Gigabytes of RAM, and a AMD Radeon 6700S GPU. BERT / DISTILBERT is not optimized for the AMD architecture, and caused massive issues during training.

Listing 2: Feature Extraction using DistilBert

```

class FeatureExtractor:
def __init__(self, model_name="distilbert-base-uncased"):
    self.tokenizer = DistilBertTokenizer.from_pretrained(model_name)
    self.model = TFDistilBertModel.from_pretrained(model_name)

def extract_features(self, books_df_processed):
    document_embeddings = []
    for author, title, desc, genres in zip(books_df_processed['author'], books_df_processed['title'], books_df_processed['description'], books_df_processed['genres']):

```

```

# Concatenate author, title, and description
input_text = author + '-' + title + '-' + desc
genre_text = '-'.join(genres)
input_text = input_text + '-' + genre_text

# Tokenize input text
inputs = self.tokenizer(input_text, padding=True,
                        truncation=True, return_tensors="tf")

# Forward pass through BERT model
outputs = self.model(inputs)

# Extract embeddings
last_hidden_states = outputs.last_hidden_state
# You can choose to use the embedding of the [CLS] token or
# pool the embeddings to get a single vector
pooled_embedding = tf.reduce_mean(last_hidden_states, axis
                                   =1)
document_embeddings.append(pooled_embedding.numpy())

# Combine document embeddings with other features
##language_features = pd.get_dummies(books_df_processed['
language_code']).values
composite_feature_vector = np.vstack([document_embeddings])

return composite_feature_vector

```

Note that `distilbert` and `distilbert_v2` are two separate models. In `v2`, we are capturing the embeddings separately for each field, and then combining them together during model formation. In the original model, we concatenate all the fields into one string, and use that to train the model. We feel that `v2` should have better performance across the board, but are unable to test it due to hardware limitations.

Word2Vec

Word2Vec is another traditional ML approach to generating embeddings. We used Word2Vec on all the available fields (title, description, author, genres), and the result was not satisfactory. The resultant model produced matches that simply matched strings within the other titles, and thus did not capture any semantic meaning behind the words themselves.

We left the model in the system, for testing and evaluation purposes.

Collaborative filtering model

We've designed a collaborative filtering model to use BERT and the ratings dataset from `goodbooks-10k`, but as we don't have very many good datapoints for user preferences, we are omitting this step from the final pipeline. As we get more and more user data, and as we get better hardware to

evaluate BERT based models, we will be able to incorporate collaborative filtering as well into our total recommender algorithm.

Listing 3: Feature Extraction using traditional Word2Vec

```
class FeatureExtractor:
def __init__(self):
    self.word2vec_model = None

def train_word2vec_model(self, books_df_processed):
    # Tokenize text (title and description) into words
    tokenized_text = [word.tokenize(title + '-' + desc) for title,
                       desc in
                           zip(books_df_processed['title'],
                               books_df_processed['description'])]

    # Train Word2Vec model
    self.word2vec_model = Word2Vec(sentences=tokenized_text,
                                    vector_size=100, window=5, min_count=1, workers=4)

def extract_features(self, books_df_processed):
    if self.word2vec_model is None:
        self.train_word2vec_model(books_df_processed)

    # Generate document embeddings using Word2Vec
    document_embeddings = []
    for title, desc in zip(books_df_processed['title'],
                          books_df_processed['description']):
        tokenized_title = word.tokenize(title)
        tokenized_desc = word.tokenize(desc)
        title_embedding = np.mean(
            [self.word2vec_model.wv.get_vector(word) for word in
              tokenized_title if word in self.word2vec_model.wv],
            axis=0)
        desc_embedding = np.mean(
            [self.word2vec_model.wv.get_vector(word) for word in
              tokenized_desc if word in self.word2vec_model.wv],
            axis=0)
        document_embeddings.append(title_embedding)

    # Binarize the genres column
    mlb = MultiLabelBinarizer()
    binarized_genres = mlb.fit_transform(books_df_processed['genres'])

    # One-hot encode the language_code
```



```
books_df_subset = pd.get_dummies(books_df_processed, columns=['
    language_code'])

# Combine document embeddings with other features
composite_feature_vector = np.hstack([binarized_genres,
    document_embeddings,
    books_df_subset.drop(columns=['genres', 'title', 'description'
    ]).values])

return composite_feature_vector
```

4.1.2 Demo Application Creation

The demo application is needed to demonstrate the functionalities of the API, and also gather user feedback for the performance of the models. The application should be easily accessible, and not weigh down the user's device much, if at all. With these considerations in mind, we chose to create a web-app to demonstrate the model technology.

Feature	Next.js	Vue.js
Framework	React-based framework for building server-side rendered (SSR) and static websites.	Progressive JavaScript framework for building interactive web interfaces.
Routing	Built-in routing capabilities using file-based routing and the 'pages' directory structure.	Vue Router is the official routing solution for Vue.js applications.
State Management	Supports various state management solutions including React Context API, Redux, and MobX.	Supports Vuex, a state management library inspired by Flux and Redux.
Server-side Rendering (SSR)	Built-in support for SSR with server-side rendering of React components.	SSR is possible using frameworks like Nuxt.js, which is built on top of Vue.js.
Community	Active community support and ecosystem with extensive documentation and resources.	Large and vibrant community with a rich ecosystem of libraries and plugins.
Learning Curve	Requires knowledge of React and JavaScript ecosystem.	Requires knowledge of Vue.js and its ecosystem.

Table 1: Comparison between Next.js and Vue.js

Our team was familiar with many web solution stacks. But overall, we wanted to keep the development within NodeJS, as our entire team was very well acquainted with JavaScript, having had a wonderful instructor to teach us. The next problem was the choice of framework. Multiple choices came up, with frontline choices being NextJS and VueJS. Given below are some notable differences between the two fullstack frameworks.

As the team member leading the development of the web application was more familiar with NextJS, we decided to choose NextJS. This coincidentally also lent very well into the nature of the application itself, as interactivity wasn't much of a priority with this application, and we wanted

to make everything fit into a simple, mostly SSR (server side rendered, for increased performance) application.

Modularity and reusability was a primary focus in the actual code itself. As we wanted the components built in this project to be used in various live projects as individual components, we felt that it was imperative that the code be clean, developer friendly, and easy to modify and extend. To this extent, we followed all the standard best practices when it comes to NextJS development. This application was built on NextJS 14, which uses a newer routing system called the App Router. There was a bit of a learning curve in understanding and the changes from older NextJS versions, but the benefits to using the newer version were enormous, especially on code organization, and on performance.

The application architecture is divvied up as follows. `page.js` contains the main SPA code, and is generated client side. NextJS by default renders everything on server side, but we had to use client side rendering in order to implement React Hooks to do state management of the data obtained from the server. We were not too concerned with performance drawbacks due to client side rendering here, as the computations performed are minimal, and deal only with computing and displaying the API data. `recommender.api.services` is the service that interacts with the API endpoint. An HTTP POST request is sent to the server, (currently hardcoded to `http://localhost:5000/predict`), and will fetch the predictions for a given book title. By default, this function requests data using the `fuzzy-tf-idf` model.

As of writing, no significant effort was made into the CSS of the web application, as we expect every actual implementation of this system to be different, and use their own individual branding. We have incorporated `react-bootstrap` to simplify the app layout, and make it look more aesthetic and user friendly for demonstration purposes.

4.1.3 Bookstore Application creation

The demo application was developed to showcase the functionalities of the API and to gather user feedback on the performance of the models. Our objectives were to create an easily accessible web application that wouldn't heavily burden the user's device. To achieve these goals, we chose to build the application using a web technology stack that our team was familiar with.

Choice of Technology Stack

Our team has extensive experience with JavaScript, making Node.js a natural choice for the backend. We aimed for a seamless development process by utilizing technologies that complement each other well. The following technologies were selected:

- **Frontend Framework:** React.js
- **Build Tool:** Vite.js
- **Styling:** Tailwind CSS
- **Database:** MongoDB Atlas
- **Backend Framework:** Node.js

Reasons for Choosing the Stack

1. **React.js:** Known for its component-based architecture, React.js allows for the creation of reusable UI components. This modularity aligns with our focus on code reusability and maintainability.
2. **Vite.js:** Vite.js offers a faster and more efficient development experience compared to traditional build tools like Webpack. Its instant hot module replacement (HMR) feature greatly enhances developer productivity.
3. **Tailwind CSS:** Tailwind CSS provides a utility-first approach to styling, enabling us to create responsive and modern UIs with minimal effort. Its integration with React.js is straightforward and enhances the overall development workflow.
4. **MongoDB Atlas:** As a fully managed cloud database service, MongoDB Atlas offers scalability, high availability, and security. It allows us to focus on application development without worrying about database management.
5. **Node.js:** Node.js is a powerful and flexible backend framework that works seamlessly with MongoDB. Our team's familiarity with JavaScript made it an ideal choice for both frontend and backend development.

Development Process

The development of the web application was led by a team member experienced with React.js. The decision to use React.js was influenced by its popularity and the availability of a large ecosystem of libraries and tools.

- **Modularity and Reusability:** The application was designed with a focus on modularity and reusability. Components were developed to be easily reused in various parts of the application.
- **Performance:** By leveraging Vite.js for the build process and React.js for the frontend, we ensured that the application was both fast and responsive.
- **Styling:** Tailwind CSS was integrated to provide a consistent and modern look and feel across the application.

Application Functionality

The demo application primarily includes two features:

1. **Home Page:** A simple landing page that provides an overview of the application and its capabilities.
2. **Show All Books:** A page that lists all the books available in the database, demonstrating the API's ability to fetch and display data.

Due to time constraints, the application does not include full CRUD (Create, Read, Update, Delete) functionalities. The current implementation focuses on demonstrating the core features of the recommendation system.

The demo application serves as a proof of concept, showcasing the potential of the API and the recommendation models. It provides a foundation for further development and enhancements based on user feedback and performance analysis data which can be later used for our proposed collaborative filtering model. By choosing a familiar and efficient technology stack, we were able to develop a robust and scalable application within a short timeframe.

4.2 Testing

Testing was performed mostly manually, and divided up into three parts. First, the API and models were tested independently, then, the Demo app's endpoints were tested, and finally, the API and the Demo App were tested together.

Stage 1 - API Testing

The API was tested rigorously using Postman to perform testing. A new workspace was created, and the local testing deployment on port 5000 was tested. Primarily, the behaviour of the API when as it handles expected and unexpected input data was analyzed.

NOTE: All the testing was done in a development environment, and no testing was done on a production server due to financial constraints of the team, and time constraints.

Test Case ID:	TC001	
Title:	Validate <code>/predict</code> route	
Precondition:	The book dataset is loaded, the model is trained and dumped, the server is running and the recommendation model is deployed on port 5000.	
Assumption:	User submits a HTTP POST request to the <code>/predict</code> endpoint, from localhost.	
Number of Test Cases	Sub Test Case	Description and Expected Results
1	TC001-1	Input a valid book title and model name and expect the system to return the top 10 recommended books.
2	TC001-2	Input only a valid book title, expects the API to return a descriptive error message.
3	TC001-3	Inputs only a invalid book title and model, expects the API to return recommendations for the closest match
4	TC001-4	Inputs a valid book title and invalid model name, expects the API to return a descriptive error message
5	TC001-4	Inputs a valid book title and valid model name and some other junk data, expects the API to return a list of recommended books

Expected Result:	<ul style="list-style-type: none"> • TC001-1: The system should return a 201 output list of 10 recommended books related to the input title. • TC001-2: The system should return a 404 JSON output with data error: Book not found! • TC001-3: The system should return a 201 output list of 10 recommended books related to the input title. • TC001-4: The system should return a 300 JSON output with data error: Invalid model! • TC001-5: The system should return a 201 output list of 10 recommended books related to the input title, ignoring the other data.
Actual Result:	<ul style="list-style-type: none"> • TC001-1: The system returned the correct list of 10 recommended books. • TC001-2: The system displayed the appropriate error message • TC001-3: The system displayed an appropriate error message. • TC001-4: The system returned the appropriate error message • TC001-5: The system returned the list of 10 recommendations.
Actual Output:	<ul style="list-style-type: none"> • TC001-1: [List of 10 recommended books] • TC001-2: [List of 10 recommended books with typo handling] • TC001-3: "Error: Book title not found."
Pass/Fail:	<ul style="list-style-type: none"> • TC001-1: Pass • TC001-2: Pass • TC001-3: Pass
Test Case ID:	TC002
Title:	User Interaction
Precondition:	The recommendation model is deployed.
Assumption:	User selects a book within the app.

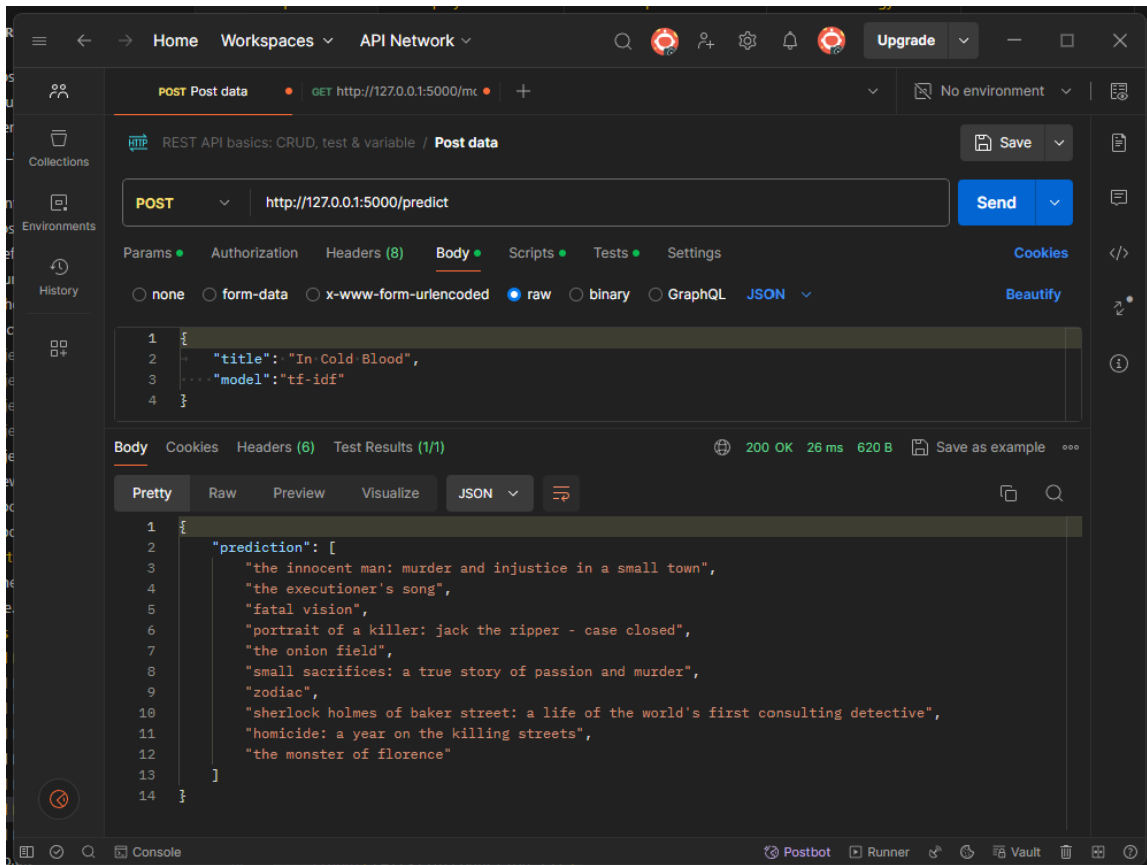


Figure 3: Test 1

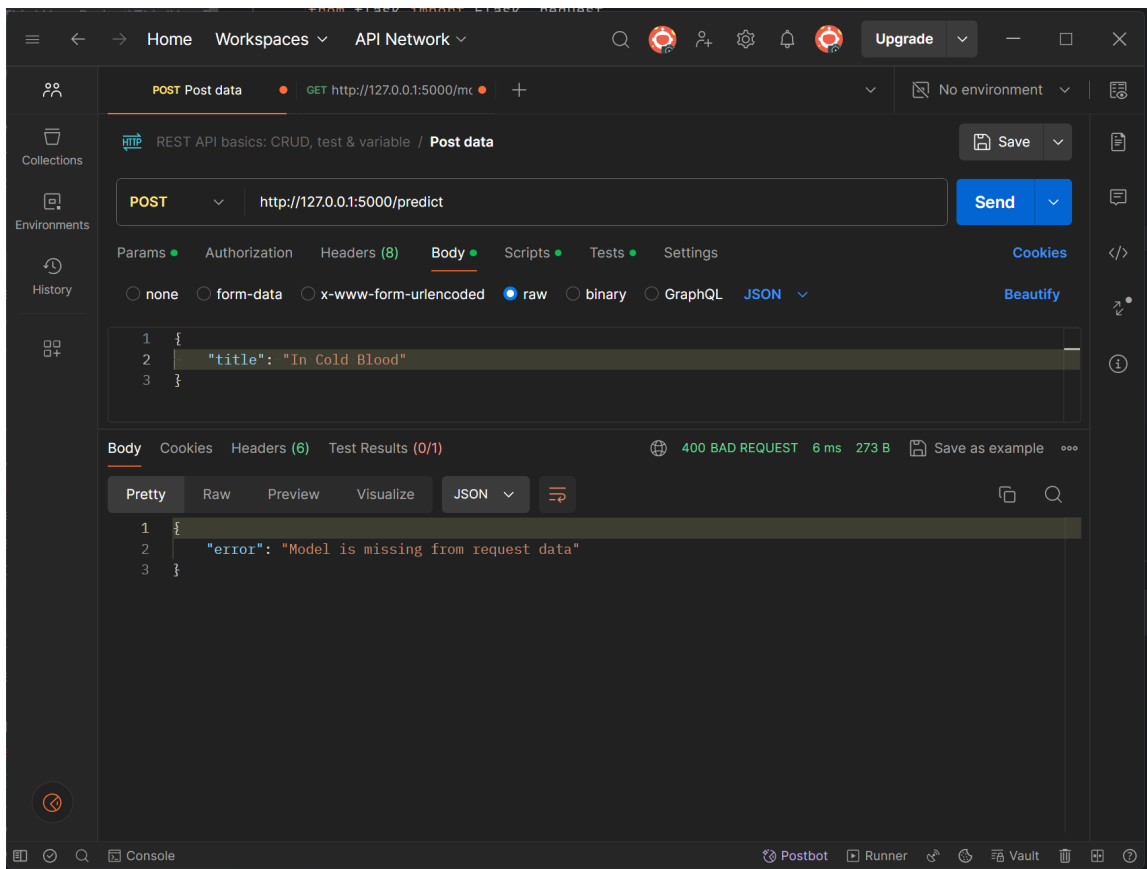


Figure 4: Test 2

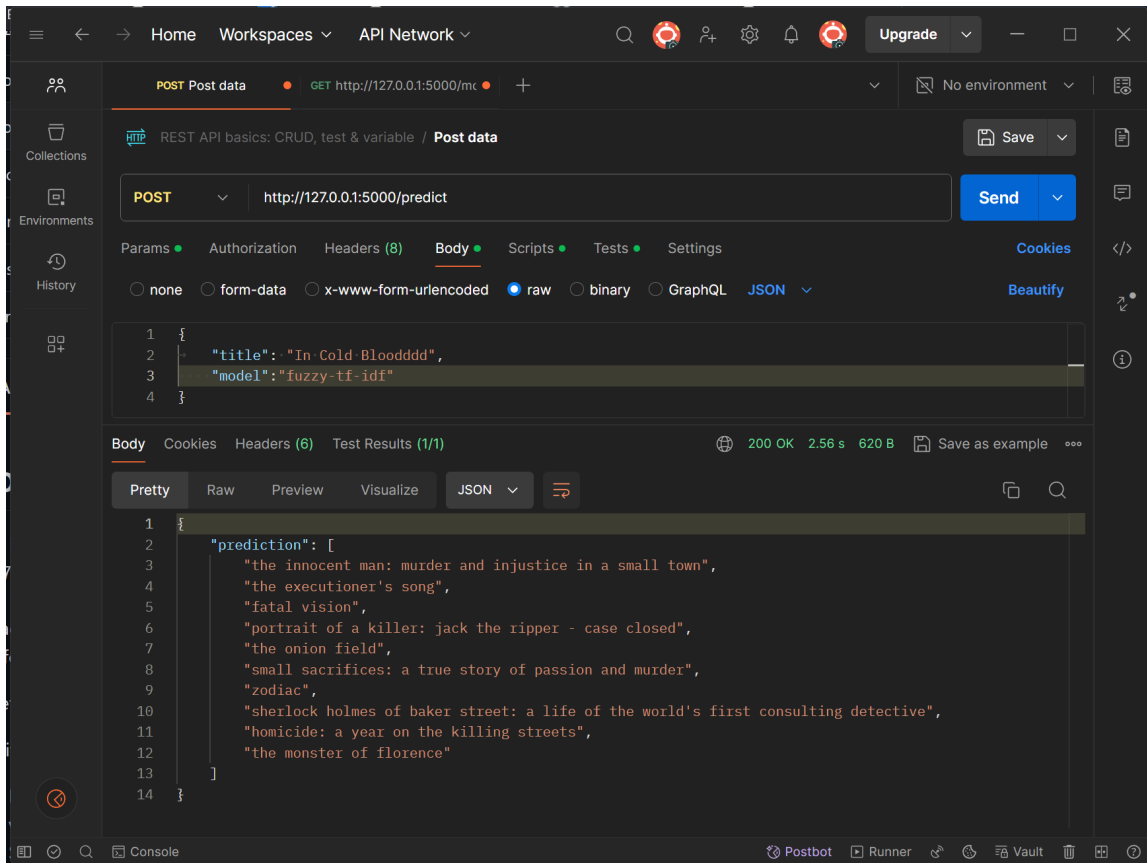


Figure 5: Test 3

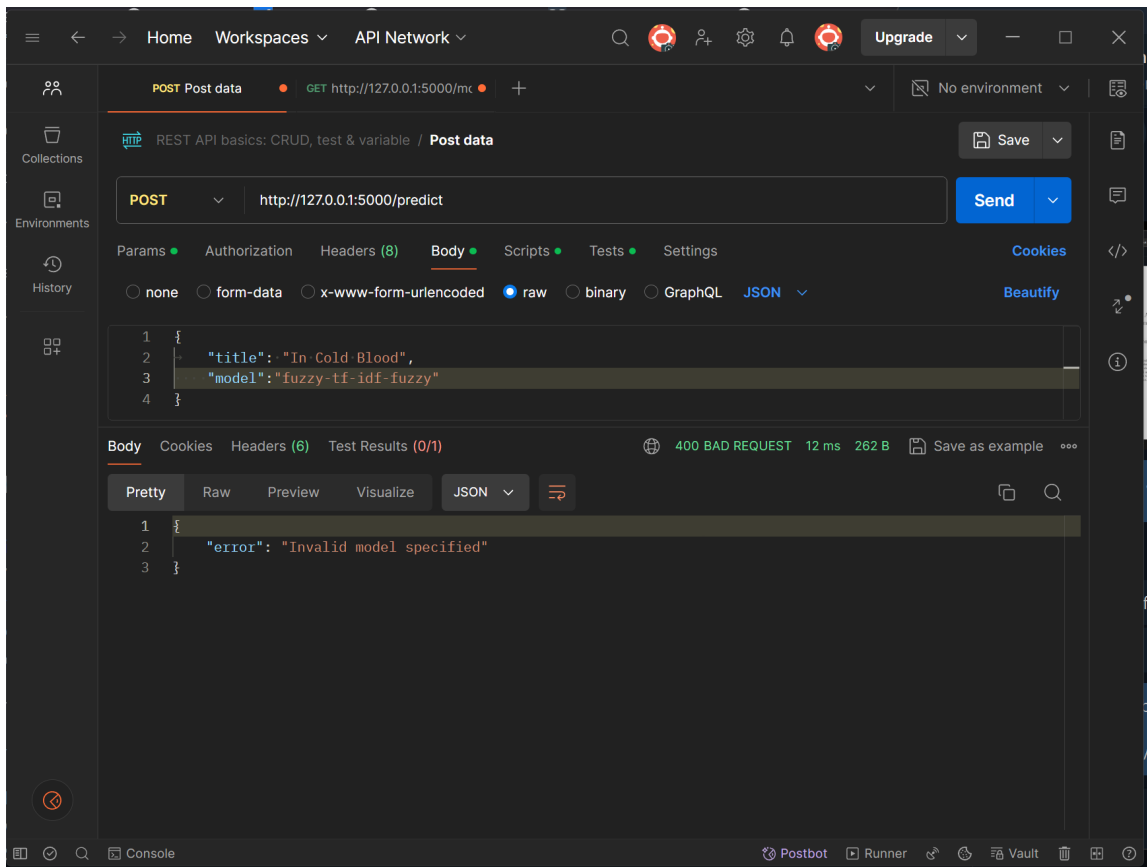


Figure 6: Test 4

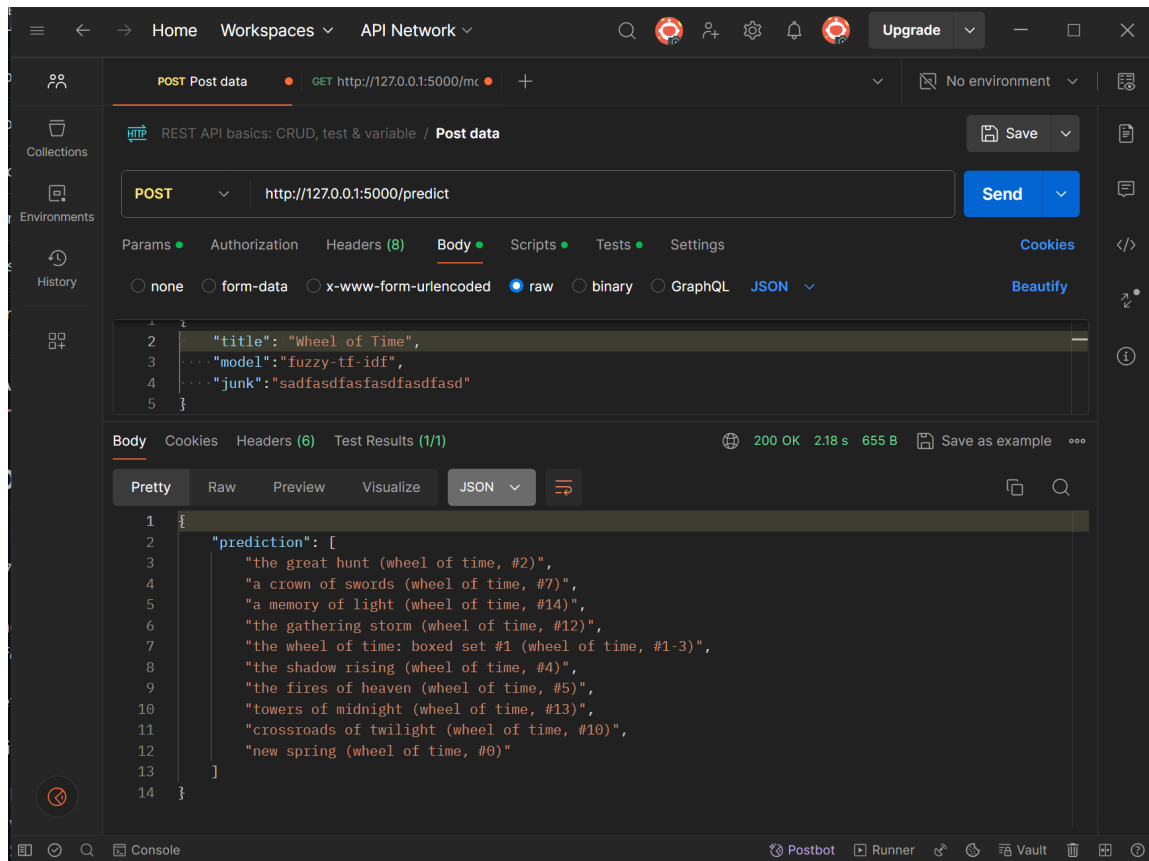


Figure 7: Test 5

Number of Test Cases	Sub Test Case	Description and Expected Results
1	TC002-1	User selects a book and provides recommendation.
2	TC002-2	App updates the interaction matrix.
Expected Result:		<ul style="list-style-type: none"> • The app should give correct results. • Subsequent recommendations consider this new interaction.
Actual Result:		<ul style="list-style-type: none"> • The app provided correct recommendations. • Subsequent recommendations considered the new interaction.
Actual Output:		<ul style="list-style-type: none"> • Correct recommendations were displayed. • Interaction matrix was successfully updated.
Pass/Fail:		<ul style="list-style-type: none"> • TC002-1: Pass • TC002-2: Pass
Test Case ID:		TC001
Title:		Get Available Models
Precondition:		The server is running and the models endpoint is accessible.
Assumption:		None
Number of Test Cases	Sub Test Case	Description and Expected Results
1	TC001-1	Send a GET request to the models endpoint with no data. Expected result: The server returns a list of available models.
2	TC001-2	Send a request other than GET to the models endpoint. Expected result: The server returns an error.
Expected Result:		<ul style="list-style-type: none"> • For TC001-1: The server returns a list of available models. • For TC001-2: The server returns an error.

Actual Result:	<ul style="list-style-type: none"> • For TC001-1: The server returned a list of available models. • For TC001-2: The server returned an error.
Actual Output:	<ul style="list-style-type: none"> • For TC001-1: List of available models: ['model1', 'model2', ...]. • For TC001-2: Error message: "Method Not Allowed".
Pass/Fail:	<ul style="list-style-type: none"> • TC001-1: Pass • TC001-2: Pass

Test Case ID:	TC002	
Title:	Book Recommendation via Next.js Application	
Precondition:	The Next.js application is running and accessible at port 3000 on the local machine.	
Assumption:	None	
Number of Test Cases	Sub Test Case	Description and Expected Results
1	TC002-1	Type in the name of a book and submit the form in the Next.js application. Expected result: The application returns a list of book recommendations.
Expected Result:	<ul style="list-style-type: none"> • For TC002-1: The Next.js application returns a list of book recommendations based on the input. 	
Actual Result:	<ul style="list-style-type: none"> • For TC002-1: The Next.js application returned a list of book recommendations as expected. 	
Actual Output:	<ul style="list-style-type: none"> • For TC002-1: List of recommended books: ['Book A', 'Book B', ...]. 	
Pass/Fail:	<ul style="list-style-type: none"> • TC002-1: Pass 	

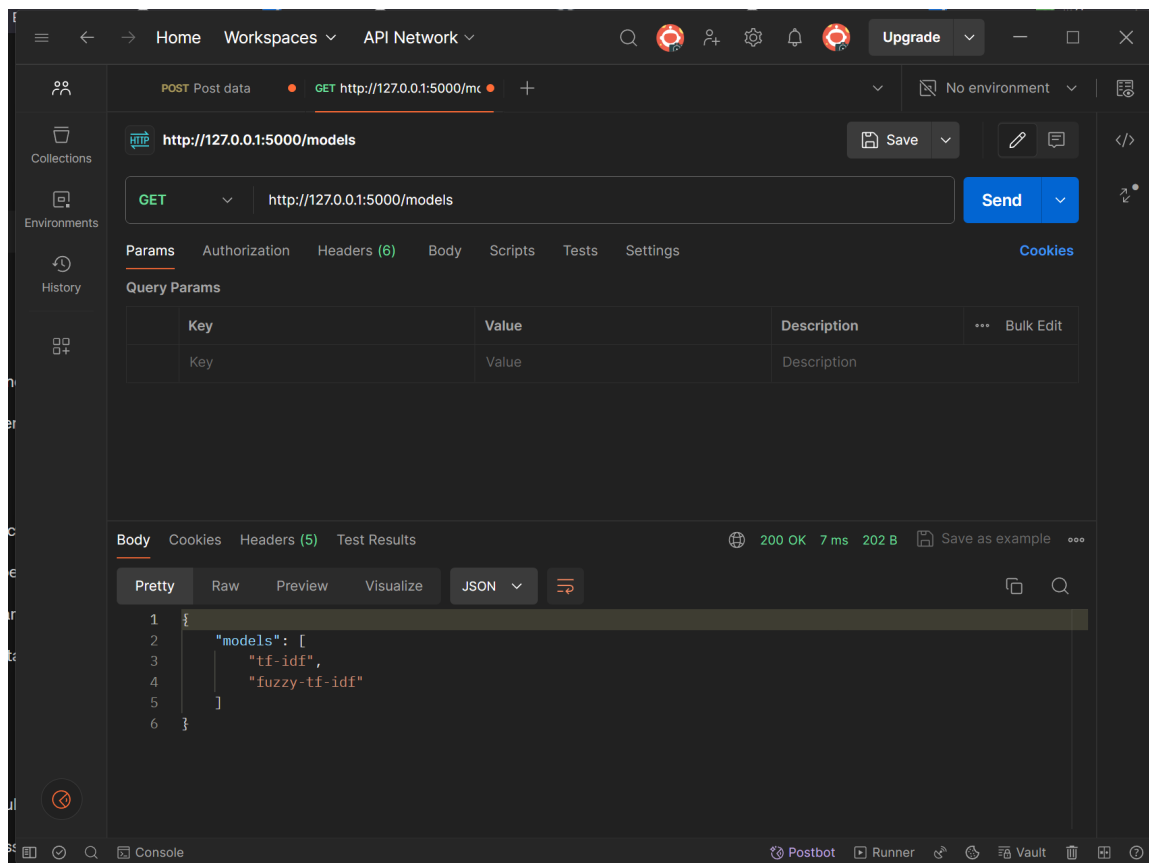


Figure 8: Test 1

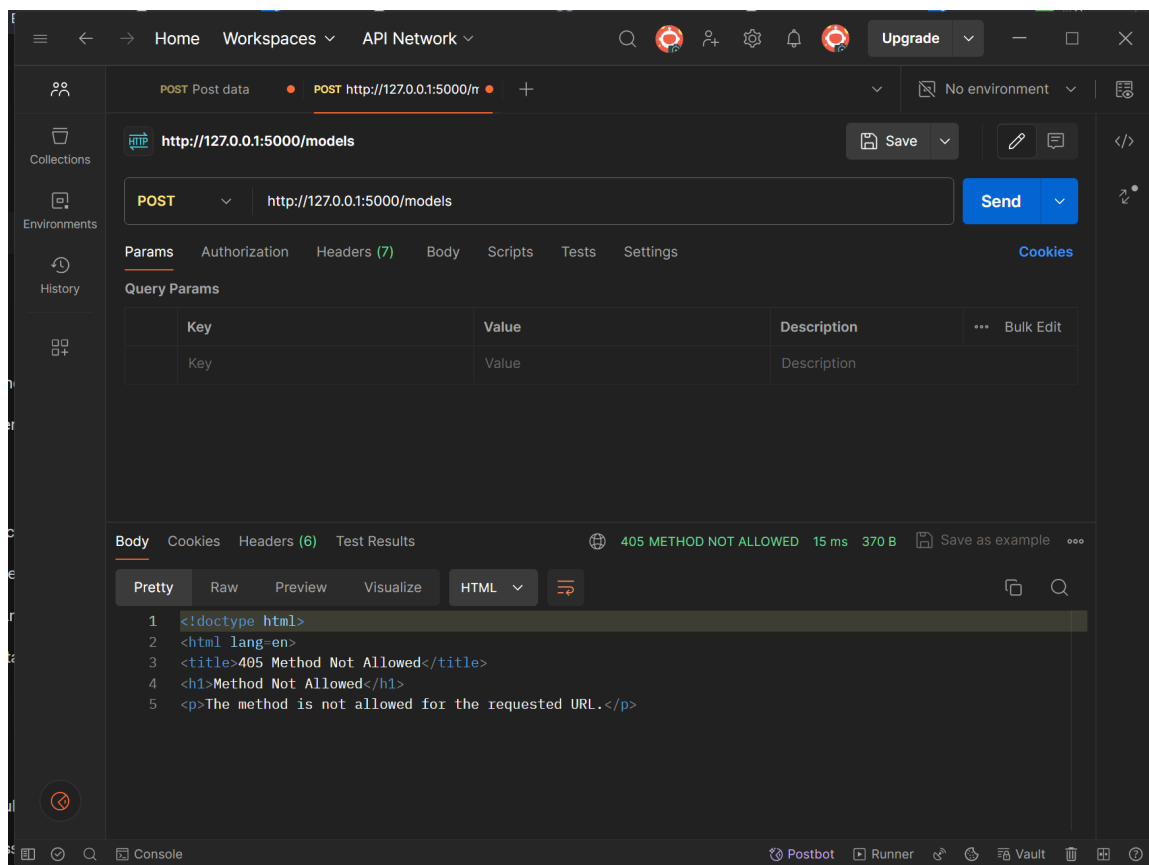


Figure 9: Test 2

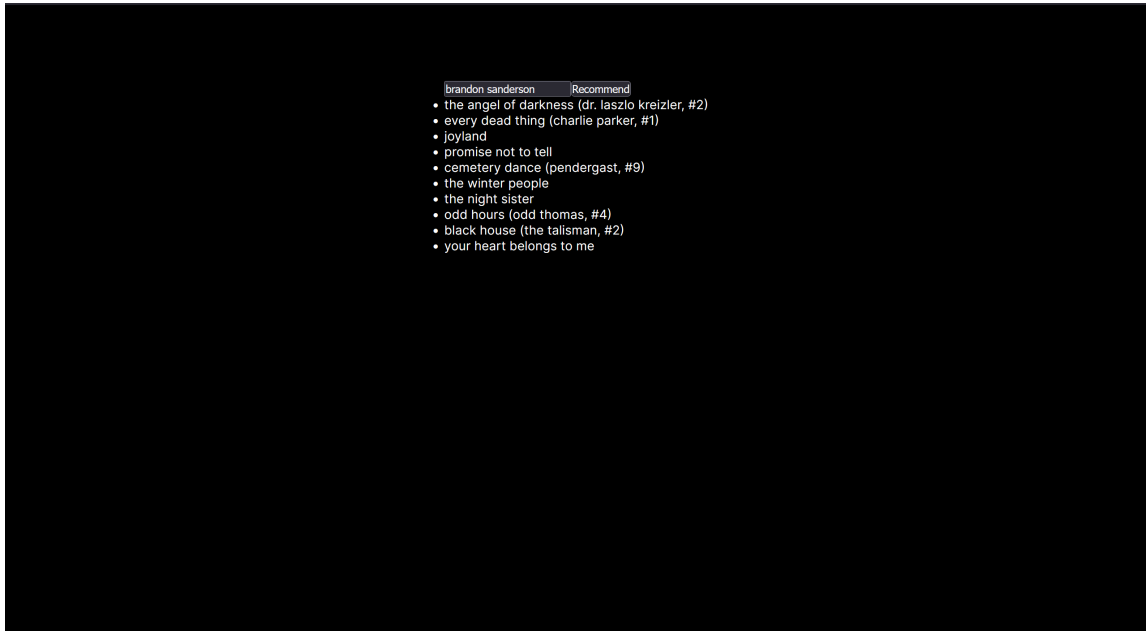


Figure 10: Rudimentary connection test

4.3 Deployment

For testing purposes, the project was deployed on the developer's local machine. Initially the application was run straight out of the IDE, but docker instructions for deploying the models were written as well, to simplify deployment in a cloud environment.

4.4 Deploying the API Server

For deploying the server on your local machine, first clone the root repository (that contains this report) to your drive. Cloning to a Windows DEV Drive is recommended to increase performance.

Ensure that Python 3.11.9 is installed on your computer. We believe that any 3.11.* version should work, but it has not been tested. Proceed at your own risk. Python 3.12 will NOT work. Don't even try.

Navigate to the `src/api-server` folder, and open a terminal there.

First, you will have to create a new virtual environment. This is to containerize your packages, so that it doesn't mess with other projects or your systemwide python installation. In this example, `testenv` is used as the environment name as this is a testing deployment.

```
python -m venv testenv
```

Next, activate the newly created venv. This script is for windows installations only.

```
.\venv\Scripts\activate
```

The activation script is different on other operating systems, please consult the documentation for venv for your operating system version to determine the correct syntax.

Then, install all necessary modules.

```
pip install -r .\requirements.txt
```

Before running the server, you can configure what models are available for use by server. Go to `app.py`, and add or remove available models from line 16 onwards. We recommend not enabling the BERT based models unless you know what you're doing, and have a powerful computer.

Finally, run the flask server.

```
flask run
```

If you wish to use Docker, a Dockerfile is available in the root directory of the api-server project. Run it on your server of choosing.

4.5 Deploying the Demo Application

For testing purposes, we recommend running the app locally. To do so, first clone the root repository (that contains this report) to your drive. As mentioned above, cloning to a Windows DEV Drive will increase compile performance.

You will need `nodejs` and `npm` installed. Please refer to the official documentation for instructions on how to install node and npm.

Start by opening up a terminal in the `src/api-demo-app` folder. Update packages by running

```
npm install
```

Next, run the development server by running,

```
npm run dev
```

Your development server is now running, and you should be able to access the page.

NOTE: The application will not function unless the api-server is also running, for obvious reasons. As far as our testing goes, we ran the api-server on port 5000, and the demo server on port 3000.

We intend to deploy the api-demo-server on Vercel, for it's ease of use and compatibility with Github for CI/CD pipelines.

Deployment is done on Vercel by first creating an account, giving conditional access to the repository to be deployed (ThirdYearProject in this case), selecting the root directory, and deploying the application. The production deployment is sourced from the main branch, and active development is conducted on other branches, and merged with the main branch.

5 Discussion

During development of this project, we faced many difficulties in implementing the initial scope of our project. The primary and insurmountable hurdle we faced was with data. In our initial proposal, we believed that we could make the recommender model multilingual, and able to understand queries from multiple languages. We primarily focussed on English, Sinhala, and Tamil as our languages of choice, as they are the main three languages of Sri Lanka.

We utilized various datasets to train our models. From datasets pre-made for the purposes of training AI, to data database dumps of popular online libraries (Osh, 2024), we had no problems

Configure Project

Project Name

third-year-project

Framework Preset

Ⓝ Next.js

Root Directory

src/api-demo-app Edit

> Build and Output Settings

> Environment Variables

Deploy

Figure 11: Settings used in deploying the application

finding ready-to-use data for training our recommender in English. The problems arose when we tried to create datasets in Sinhala and Tamil languages.

Towards this end, we requested the assistance of the National Library of Sri Lanka, and obtained their Sri Lanka National Bibliography (Sri Lanka National Library, 2000–2022) (Data from January 2000, to June 2022). This is a compendium of 22 years of every book published in Sri Lanka, and the data was available trilingually. The majority of the data is in digitally typed PDFs, while some of the older SLNB volumes are Scanned PDFs of the original print volumes. Even with the digital PDFs, the bibliography entries are incredibly hard to parse.

Critical perspectives on open development :
 empirical interrogation of theory
 construction / ed. Arul Chib, Caitlin M.
 Bentley and Matthew L. Smith. - England :
 The MIT Press, 2020. - xi,302 p. ; 23 cm. -
 (The MIT press - international development
 research centre series ;)
 Ad-Bc : Unpriced (1096164 NL)
 ISBN 978-0-262-54232-6

1. Economic development
2. Information commons
3. Information society
4. Information technology-Social aspects
5. Open source software
6. Sri Lanka Standard Institution

Figure 12: A page of bibliographies

As you can see from the above bibliography page, the bibliographies are given in a two column layout, with the bibliographies themselves having no consistent format.

The problem is made worse by the fact that, with the way the PDF is made, the two column layout is not parsed properly by pdf analysis algorithms. This makes it such that data from two colums are mashed up together, creating an unreadable jumble of text. In order to utilize this resource, a massive undertaking to extract, clean, and standardize the text would have to be attempted.

Upon research, we believe that mass scale datasets used by companies such as OpenAI, Google, Amazon, and Microsoft are mostly created by hand. A novel technique for utilizing an AI preprocessor model to create structured data from internet scale mass data was proposed by OpenAI, and we feel that the approach used there (manually generate 20% of the data, have the preprocessor model extract the rest) would have been effective, but hardware constraints limited us in the exploration of this technique. The technique proposed by OpenAI (Baker et al., 2022) leverages a structured dataset which is used to train a model to extract data from internet-scale data. This data is obtained by mass scale human workers, obtained through the Amazon M-Turk program. All in all, the paper estimates that they spent roughly 10,000 US Dollars to extract 70,000 points of data. That computes to around Rs.3,000,000, or 3 Million Rupees.

SRI LANKA NATIONAL BIBLIOGRAPHY

SUBJECT SECTION

0 0 0 COMPUTER SCIENCE, INFORMATION, GENERAL WORKS

001.94 - Mysteries

Razeen, Muhammad
The wonders of the world / Muhammad
Razeen ; tr. by A. C. M. Wabanbey. -
Kurunegala : Author, 2019. - 56 p. :
photos ; 21 cm.
Ad-Bc : Rs. 350.00 (1097524 NL)
ISBN 978-955-43511-7-2

1. Curiosities and wonders

2 0 0 RELIGION

2 9 0 OTHER RELIGIONS

294.3435095493 - Sri Lanka

Somasundara, J. W. D.
The most sacred Sri Pada / J. W. D.
Somasundara and H. M. Jayantha
Wijeratna ; tr. by Sunil Wijeyesinghe. -
Colombo : S. Godage, 2022. - 128 p. :
photos ; 22 cm.
Ad-Bc : Rs. 950.00 (1104779 NL)
ISBN 978-624-00-1311-6 (477668 NA)

1. Adam's Peak (Sri Lanka)
2. Mountains-Sri Lanka

3 0 0 SOCIAL SCIENCES

303.4833 - Communication

Critical perspectives on open development :
empirical interrogation of theory
construction / ed. Arul Chib, Caitlin M.
Bentley and Matthew L. Smith. - England :
The MIT Press, 2020. - xi,302 p. ; 23 cm. -
(The MIT press - international development
research centre series ;)
Ad-Bc : Unpriced (1096164 NL)
ISBN 978-0-262-54232-6

1. Economic development
2. Information commons
3. Information society
4. Information technology-Social aspects
5. Open source software
6. Sri Lanka Standard Institution

305.89481105493 - Sri Lanka

Up country Tamils : charting a new future in
Sri Lanka / ed. Daniel Bass and B.
Skanthakumar. - Colombo : International
Centre for Ethnic Studies, 2019. -
xxi,243 p. ; 22 cm.
Ts-Cbc : Unpriced (1084209 NL)
ISBN 978-955-580-237-6

1. Malaiyaha Tamil (Sri Lankan people)-
Sri Lanka
2. Race relations-Political aspects
3. Sri Lanka-Ethnic relations
4. Sri Lanka-Politics and government

3 2 0 POLITICAL SCIENCE

320.95493 - Sri Lanka

Perera, L. C.
Facing our own citizens in conflict
situations : the Southern JVP insurrection
1987 - 1989 / L. C. Perera. - [s. l.] : [s. n.],
2022. - 336 p. ; 21 cm.
Ad-Bc : Rs. 1000.00 (1098304 NL)
ISBN 978-624-5771-06-6 (475457 NA)

1. Janatha Vimukthi Peramuna
2. Sri Lanka-Politics and government

Figure 13: A page of bibliographies

A large scale orchestrated effort to parse this data into a machine readable format would prove useful for further research into LLMs and Deep Neural networks which can parse and understand Sinhala and Tamil languages, but is beyond the scope of this project at this time.

While our initial goal was to extract this data and store it in a immutable datastore like Blockchain or IPFS, we were ultimately unable to do so due to the aforementioned issues with data extraction. We hope that our work so far will prove useful to researchers and developers in the future hoping to work on this same matter.

Further, we faced a plethora of issues when it comes to collaboration. Due to our relative inexperience with Git, managing the various branches, tracking, and updating them, and merging changes was handled haphazardly. The availability of `git cherry-pick` was sadly realized a very late stage in the project, and this caused a few members contributions to not be recorded in Github's statistics, as some code was overwritten and files moved around in the final rush to organize and present the code.

Further Progress

We do believe that our work has many avenues for improvement and further development. With better hardware, we believe that the BERT based ML Recommender model can be deployed successfully, and the DistilBert model could be used in a production environment without too much hassle.

Further, we believe that utilizing manpower to parse and analyze the SLNB data would prove a very fruitful endeavour. The data can be used for applications well beyond what is in this scope of this Software Development project, and will serve our mother country well.

It is also our aim to package the api server into a ready-to-go Docker Image. A docker image is already provided with the project files, but we feel that the API server could be improved further with more modular design, and with the option of adding many more models, and even using a bring-your-own-model sort of system. As long as the recommendation, user data, and metadata formats remain consistent, all deployments can talk to each other and exchange data to boost productivity.

6 Conclusion

In conclusion, we believe that we have done a solid attempt at researching the difficulties posed by creating recommendation algorithms, and the unique challenges of making it work in a multilingual environment like Sri Lanka.

We are also releasing our work under the permissive MIT License, allowing everyone to use our work freely, extend, modify, use, and abuse it to it's limits, with the hopes that this will improve the knowledge availability and literacy of our country, and the availability of solid research and data on utilizing our native languages in AI/ML contexts.

What data we do have, we have uploaded to a IPFS node, making it freely available for everyone to use, and this data will last throughout our lifetimes and beyond as it is a decentralized repository, similar in nature to LibGen.

References

- Baker, B., Akkaya, I., Zhokhov, P., Huizinga, J., Tang, J., Ecoffet, A., . . . Clune, J. (2022, Jun). Video pretraining (vpt): Learning to act by watching unlabeled online videos. *arXiv preprint arXiv:2206.11795*. Retrieved from <https://arxiv.org/abs/2206.11795> doi: 10.48550/arXiv.2206.11795
- Osh, M. (2024). *Goodbooks-10k-extended*. <https://github.com/malcolmosh/goodbooks-10k-extended.git>. (Accessed: 2024)
- Sri Lanka National Library. (2000–2022). *Sri lanka national bibliography*. Sri Lanka National Library. (Accessed: 2023)