SMART BOOK RECOMMENDATION SYSTEM USING MACHINE LEARNING

A PROJECT REPORT

Submitted by

FAISAL AHAMAD M(811721243016)

MOHAMMED FAIZAL T(81172143033)

MOHAMMED IMRAN KHAN M(811721243034)

YOGESH D(811720243305)

in partial fulfilment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

K.RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

SAMAYAPURAM – 621 112

MAY, 2025

K.RAMAKRISHNAN COLLEGE OF TECHNOLOGY
(AUTONOMOUS)
SAMAYAPURAM – 621 112

BONAFIDE CERTIFICATE

Certified that this project report titled **"SMART BOOK RECOMMENDATION SYSTEM USING MACHINE LEARNING"** is the bonafide work of **FAISAL AHAMAD M (REG.NO: 811721243016), MOHAMMED FAIZAL T (REG. NO: 81172143033), MOHAMMED IMRAN KHAN M(REG. NO: 811721243034), YOGESH D (REG. NO: 811720243305)** who carried out the project under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. T.Avudaiappan M.E., Ph.D.,
HEAD OF THE DEPARTMENT

Department of Artificial Intelligence
K.Ramakrishnan College of Technology
(Autonomous)
Samayapuram – 621 112

SIGNATURE

Mr. R. Roshan Joshua, M.E.,
SUPERVISOR

ASSISTANT PROFESSOR
Department of Artificial Intelligence
K.Ramakrishnan College of Technology
(Autonomous)
Samayapuram – 621 112

Submitted for the viva-voce examination held on ……

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION

We jointly declare that the project report on **"SMART BOOK RECOMMENDATION SYSTEM USING MACHINE LEARNING"** is the result of original work done by us and best of our knowledge, similar work has not been submitted to **"ANNA UNIVERSITY CHENNAI"** for the requirement of Degree of **BACHELOR OF TECHNOLOGY**. This project report is submitted on the partial fulfilment of the requirement of the award of Degree of **BACHELOR OF TECHNOLOGY**.

Signature

_____

FAISAL AHAMAD M

_____

MOHAMMED FAIZAL T

_____

MOHAMMED IMRAN KHAN M

_____

YOGESH D

Place: Samayapuram

Date:

ACKNOWLEDGEMENT

It is with great pride that we express our gratitude and in-debt to our institution "**K.Ramakrishnan College of Technology (Autonomous)**", for providing us with the opportunity to do this project.

We are glad to credit honourable chairman **Dr. K.RAMAKRISHNAN**, **B.E.,** for having provided for the facilities during the course of our study in college.

We would like to express our sincere thanks to our beloved Executive Director **Dr. S. KUPPUSAMY, MBA, Ph.D.,** for forwarding to our project and offering adequate duration in completing our project.

We would like to thank **Dr. N. VASUDEVAN, M.E., Ph.D.,** Principal, who gave opportunity to frame the project the full satisfaction.

We whole heartily thank to **Dr. T.AVUDAIAPPAN**, **M.E., Ph.D.,** Head of the department, **ARTIFICIAL INTELLIGENCE** for providing his encourage pursuing this project.

We express our deep and sincere gratitude to our project guide **Mr. R. ROSHAN JOSHUA**, **M.E.,** Department of **ARTIFICIAL INTELLIGENCE,** for his incalculable suggestions, creativity, assistance and patience which motivated us to carry out this project.

We render our sincere thanks to Course Coordinator and other staff members for providing valuable information during the course.

We wish to express our special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

ABSTRACT

The rapid growth of digital literature and online reading platforms has created a need for intelligent systems that help users discover relevant books efficiently. This project, titled "Smart Book Recommendation System Using Machine Learning," aims to provide personalized book suggestions based on user preferences and book content. The system uses a hybrid recommendation approach, combining Content-Based Filtering and Collaborative Filtering to improve accuracy and diversity in recommendations. Textual features such as book titles, authors, genres, and descriptions are processed using TF-IDF vectorization and cosine similarity to measure content closeness, while user behavior is analyzed using a user-item rating matrix. The backend is developed in Python, with data handling done using pandas, and the machine learning logic implemented using scikit-learn. A user-friendly Streamlit web interface allows users to input book names and receive instant recommendations. Additionally, a chatbot powered by OpenAI or Chatbase enhances interactivity.

.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

API - Application Programming Interface
CLI - Comment Line Interface
KNN - K-Nearest Neighbour
LSTM - Long Short-Term Memory
MAE - Mean Absolute Error
OS - Operating System
RMSE - Root Mean Square Error
RPM - Revolutions Per Minute
SVM - Support Vector Machine

CHAPTER 1
INTRODUCTION

In today's information-saturated world, finding the right book often becomes a daunting task, with readers overwhelmed by an endless sea of options. The "Smart Book Recommendation System Using Machine Learning" project seeks to address this challenge by transforming historical book data from Kaggle into a dynamic and intelligent recommendation engine. By harnessing the power of Python within a Jupyter Notebook environment and utilizing the pandas library for meticulous data manipulation, this project lays a solid foundation for capturing the intricate preferences of individual readers. Central to the system is the application of cosine similarity, a mathematical technique that quantifies the closeness between books, enabling the engine to generate hyper-personalized recommendations that truly resonate with each user's unique taste. To ensure the model remains robust and easily deployable, the pickle library is used for model persistence. Enhancing user engagement further, the project features an interactive web application built with Streamlit, which offers a seamless interface where readers can simply enter their favorite book details to receive curated suggestions. Complementing these components is a fully customized ChatBot powered by the OpenAI API, designed exclusively to handle book-related queries with insightful and context-aware responses. Together, these elements converge to create a revolutionary system that not only simplifies literary discovery but also enriches the reading experience by making personalized recommendations both intuitive and engaging.

1.1 BACKGROUND

The exponential growth of digital content in today's online world has dramatically transformed how we access and consume information, making the task of finding the right book increasingly challenging. Traditional methods of book discovery, such as browsing physical shelves or static online catalogs, often fall short in addressing the diverse and ever-evolving tastes of modern readers. This context has led to the rise of personalized recommendation systems, which leverage advanced data science and machine learning techniques to offer tailored suggestions. Our project, the Smart Book Recommendation System Using Machine Learning, builds on this foundation by utilizing historical book data

sourced from Kaggle. By processing and analyzing this vast dataset using Python and the pandas library within a Jupyter Notebook environment, the system extracts meaningful patterns and relationships between literary works. The use of cosine similarity further refines these insights by quantifying the closeness between books, while model persistence is achieved through the pickle library to ensure reliability and scalability. Moreover, an interactive web application developed with Streamlit, along with a dedicated ChatBot powered by the OpenAI API, enriches the user experience, transforming the process of literary discovery into an engaging and intuitive journey.

1.2 PROBLEM STATEMENT

In today's digital era, the sheer volume of books available online creates an overwhelming challenge for readers seeking personalized and relevant recommendations. Traditional discovery methods, such as static catalogs and generic search functions, often fail to capture individual tastes, resulting in suggestions that do not truly resonate with the reader's unique preferences. This problem is further compounded by the sparse and heterogeneous nature of available data, which makes it difficult for conventional systems to accurately model and predict user interests. Consequently, readers are left sifting through a vast array of options without effective guidance, while existing recommendation systems often lack interactive, real-time engagement and fail to adapt to the nuanced needs of each user. The core challenge, therefore, is to develop a smart book recommendation system using machine learning that can intelligently analyze historical book data from Kaggle, leverage cosine similarity for precise relationship mapping, ensure robustness through model persistence with the pickle library, and enhance user experience via an interactive Streamlit web application coupled with a dedicated ChatBot powered by the OpenAI API.

1.3 OBJECTIVES

- Develop a robust data preprocessing pipeline to clean and transform historical book data sourced from Kaggle using Python and pandas.
- Implement advanced feature engineering to extract meaningful attributes from raw

book data that enhance recommendation accuracy.

- Utilize cosine similarity to accurately measure the closeness between books, thereby capturing intricate relationships among literary works.
- Design and train machine learning models to predict user preferences based on historical ratings and interactions.
- Ensure model resilience and scalability by employing the pickle library for efficient model persistence and future deployment.
- Create an interactive web application using Streamlit that provides an intuitive interface for users to input preferences and receive personalized recommendations.
- Integrate a fully customized ChatBot powered by the OpenAI API to offer dynamic, book-centric conversational support and enhance user engagement.
- Conduct thorough evaluations and testing to refine the system's performance, ensuring that the recommendations are both accurate and aligned with individual user tastes.

CHAPTER 2

LITERATURE SURVEY

2.1 BOOK RECOMMENDING USING TEXT CATEGORIZATION WITH EXTRACTED INFORMATION

Raymond J Mooney, Paul N. Bennett, Loriene Roy

Content-based recommender systems suggest docu ments, items, and services to users based on learning a profile of the user from rated examples containing information about the given items. Text categoriza tion methods axe very useful for this task but generally rely on unstructured text. We have developed a book recommending system that utilizes semi-structured in formation about items gathered from the web using simple information extraction techniques. Initial ex perimental results demonstrate that this approach can produce fairly accurate recommendations.

Merits

Integration of Information Extraction, Effective Use of ML Algorithms.

Demerits

Lack of User Interaction & Dynamic Adaptation, Absence of Hybrid Approach.

2.2 BOOK RECOMMENDATION SYSTEM USING COLLABORATIVE-FILTERING SYSTEM

Arthur C Augustian, Gloriya Mathew.

Book recommendation system here uses collaborative filtering approach to recommend books of similar types. Collaborative filtering is an approach to build recommendation engines which is widely used in data mining and information retrieval. The recommendation system in the proposed system uses Memory based collaborative filtering. Based on a collection of user preferences, memory-based collaborative filtering (CF) creates suggestions for objects. This strategy is based on the notion that an active user's interests will be more likely to match those of other users who share similar preferences as the active user.

Merits

Efficient Use of Collaborative Filtering, Scalability and Performance.

Demerits

Data Sparsity Issue, No Use of Deep Learning or Advanced Machine Learning.

2.3 BOOK RECOMMENDATION SYSTEM USING MACHINE LEARNING

S. R. Hiray, Atish Bhosale, Komal Patil, Amruta Gaikwad

Users can use book recommendation systems to search and select books from a number of options available on the web or elsewhere electronic sources. They give the user a little bit selection of products that fit the description, given a large group of objects and a description of the user needs. Our system will simply provide recommendations. Recommendations are based on previous user activity, such as purchase, habits, reviews, and likes. These systems gain lot of interest. In the proposed system, we have a big problem: when the user buys book, we want to recommend some books that the user can enjoy. Buyers also have a great deal of options when it comes to recommending the best and most appropriate books for them. User development privacy while placing small and minor losses of accuracy. Recommendations. The proposed recommendation system will provide user's ability to view and search the publications and using the Support Vector Machine (SVM), will list the most purchased and top rated books based on the subject name given as input.

Merits

Use of SVM, Flexible and Scalable, Automated Book Selection Process.

Demerits

Limited Content-Baseed Filtering, Lack of Real-World deployment.

2.4 ENHANCING PERSONALIZED BOOK RECOMMENDER SYSTEM

Abdulgafar usman, Abubakar roko, Aminu B. Abba Almu.

Recommender systems (Rs) are widely used to provide recommendations for a collection of items or products that may be of interest to a user or a group of users. Because of its superior performance, Content-Based Filtering (CBF) is one of the approaches that are commonly utilized in real-world Rs using Term Frequency and Inverse Document Frequency (TF-IDF) to calculate document similarities. However, it computes document similarity directly in the word-count space. We propose a user-based collaborative filtering (UBCF) method to solve the problem of limited content analysis, which leads to a low prediction rate for large vocabulary. In this study, we present an algorithm that utilizes the Euclidean distance similarity function to solve the identified problem. The performance of the proposed scheme was evaluated against the benchmark scheme using different performance metrics. The proposed scheme was implemented and experimentally tested using benchmark datasets (Amazon review datasets). The results revealed that the proposed scheme achieved better performance than the existing recommender system in terms of Root Mean Square Error (RMSE), reducing the errors by 29%, and also increasing the Precision and Recall by 51.4% and 55.8% respectively in the 1 million datasets.

Merits

Improved Accuracy with Hybrid Filtering, Use of large scale Dataset.

Demerits

Limited Feature extraction from Book content, High Computational Task.

2.5 MACHINE LEARNING-BASED BOOK RECOMMENDER SYSTEM

kalid Anwar, Jamshed Siddiqui, Shabab Saquib Sohail.

In this paper, we present the first results of the ACT@HOME research project which aims to develop an an artificially intelligent virtual assistant (VA) to engage and help older adults with Alzheimer's disease (AD) to complete activities of daily living (ADL) more independently. In order to define the most appropriate prompting style for each user profile, we performed 12 semi structured qualitative interviews with dyads of elderly care home residents and their family caregiver. During these interviews, we presented the virtual assistant and the different 'static' prompts to support people in the activity of hand washing. We gathered as much feedback and suggestions as possible coming directly from the end users about how to improve the provided prompts and thus, increase acceptability. The results are presented around three extracted themes: a) comments on current design of the virtual assistant, b) perceived usefulness, user adoption and c) suggestions for improvements. These should guide in the future developers of assistive technology to support elderly care home residents.

Merits

Comprehensive survey of ML Techniques, Comparison

of Evaluation Metrics.

Demerits

Lack of experimental implementation, no user study or real-world

validation.

CHAPTER 3
SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Traditional book recommendation systems primarily rely on basic filtering methods that utilize popularity metrics or rudimentary collaborative filtering techniques. These conventional systems often operate using static catalogs and general user trends, which tend to offer generic recommendations that do not fully account for the unique tastes and preferences of individual readers. While major platforms like Amazon or Goodreads employ more advanced algorithms, many existing systems still struggle with issues such as data sparsity, inadequate personalization, and limited interactivity. In most cases, the underlying algorithms use simplistic similarity measures that fail to capture the nuanced relationships between literary works, leading to recommendations that can feel impersonal and repetitive. Moreover, many of these systems do not integrate modern features like real-time user interaction or conversational interfaces, which could otherwise enhance the user experience and provide a more engaging way to explore book suggestions. As a result, there is a growing need for a smart, machine learning-based recommendation engine that not only leverages historical book data effectively but also incorporates advanced techniques to deliver highly personalized and interactive recommendations

Algorithm Used

- **Collaborative Filtering**

  Collaborative filtering is a widely used algorithm in existing book recommendation systems. It works by finding patterns based on user-item interactions (such as ratings or reviews). It assumes that users who have agreed in the past will agree again in the future. However, it suffers from the cold-start problem and struggles with sparse datasets.

- **Content-Based Filtering**

  This algorithm recommends books similar to those the user has previously liked, based on features like genre, author, or keywords. It builds a profile for each user and item, but it can be limited by a narrow range of suggestions and

cannot exploit the tastes of other users.

- **Popularity-Based Recommendations**

  These systems simply recommend the most popular or highest-rated books. While easy to implement, they lack personalization and fail to cater to unique user preferences.

- **Item-Based Similarity (Cosine Similarity)**

  Some traditional systems use cosine similarity to measure the closeness between books based on user ratings or metadata. Although it improves similarity-based recommendations, it doesn’t adapt well to dynamic changes in user behavior.

- **User-Based Similarity**

  This approach finds users who have similar preferences and recommends books liked by them. It’s often inaccurate when the user base is large or when few interactions are available for new users.

- **Matrix Factorization (Basic SVD)**

  Some existing systems implement simple Singular Value Decomposition (SVD) to reduce dimensionality and uncover latent relationships. While effective to an extent, they often lack scalability and adaptability without deeper machine learning enhancements.

- **Heuristic Filters**

  Heuristic techniques are used for filtering recommendations based on basic rules like minimum ratings, genre restrictions, or age preferences. These are static and do not offer learning or adaptability.

- **Manual Curation**

  In some platforms, part of the recommendation is supplemented by manually curated book lists or editor's picks. This lacks automation and does not reflect personalized interests of users.

3.1.1 Drawbacks

- Minimal personalization capability.
- Struggles with data sparsity.
- Cold-start problem persists.
- Limited context comprehension.
- Static recommendation logic
- Over-reliance on popularity
- Inflexible algorithm parameters
- Insufficient dynamic learning
- Poor scalability with growth
- Lack of adaptive feedback

3.2 PROPOSED SYSTEM

The proposed system is a smart book recommendation platform designed to overcome the limitations of traditional methods by delivering hyper-personalized, context-aware suggestions. At its core, the system leverages historical book data from Kaggle, which is meticulously preprocessed and analyzed using Python, pandas, and Jupyter Notebook. By applying cosine similarity, the system effectively quantifies the relationships between books, capturing the nuanced preferences of individual readers. To ensure robustness and facilitate future scalability, the trained model is persisted using the pickle library.

Enhancing user engagement, the system features an interactive web application developed with Streamlit, providing an intuitive interface where users can effortlessly input their favorite book details and receive real-time recommendations. Furthermore, the integration of a fully customized ChatBot powered by the OpenAI API offers dynamic, book-centric conversational support, enabling the system to respond to queries and refine recommendations based on user interactions. This proposed approach not only addresses common issues like data sparsity and cold-start problems but also introduces adaptive learning mechanisms to deliver a continuously improving, personalized literary discovery experience.

Algorithm Used

- **Data Preprocessing and Feature Engineering**

  The system begins by cleaning and transforming historical book data from Kaggle using Python and pandas. This process includes filtering out invalid entries, handling missing values, and extracting relevant features from book metadata to create a structured dataset for model training.

- **Cosine Similarity for Content-Based Filtering**

  - Central to the recommendation engine is the use of cosine similarity. Books are represented as feature vectors (derived from genres, ratings, authorship, etc.), and cosine similarity is computed to measure the closeness between these vectors. This technique allows the system to identify and recommend books that are similar in content to those a user has already enjoyed. Tokenization: Breaking

down text into individual words or tokens.

- Cosine similarity is inherently insensitive to the magnitude of feature vectors, ensuring that differences in scale or volume among book features do not bias the similarity calculation. Named Entity Recognition (NER): Identifying named entities such as names, locations, organizations, etc.

- This method efficiently handles high-dimensional feature representations, making it well-suited for processing complex datasets with numerous attributes.

- **Collaborative Filtering**

In addition to content-based recommendations, the system employs collaborative filtering to leverage user behavior. By analyzing patterns in user ratings and interactions, the algorithm identifies similar users or books and predicts a user's potential interest in unseen titles. This method dynamically refines recommendations based on collective user insights.

- **Model Persistence Using Pickle**

To ensure the system is both robust and scalable, the trained recommendation model is serialized and saved using the pickle library. This approach allows the model to be quickly reloaded for real-time predictions without requiring retraining, thereby enhancing deployment efficiency

- **Interactive Web Application with Streamlit**

An integral component of the proposed system is the interactive web interface developed with Streamlit. Although not a machine learning algorithm, this module is critical for user engagement—it captures user inputs, displays personalized recommendations, and enables dynamic interaction with the underlying recommendation engine.

- **ChatBot Integration via OpenAI API**

  To further enrich the user experience, a dedicated ChatBot is integrated into the system. Powered by the OpenAI API, the ChatBot handles natural language queries, provides additional context about recommendations, and engages users in meaningful conversations, enhancing the overall interactivity of the system.

- **Evaluation Metrics and Feedback Loop**

  The system employs evaluation metrics such as Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) to assess the accuracy of the recommendations. A continuous feedback loop monitors user interactions, enabling adaptive learning that incrementally improves the model's predictive capabilities.

- **Hybrid Recommendation Approach**

  By combining both content-based filtering (via cosine similarity) and collaborative filtering techniques, the proposed system leverages the strengths of each method. This hybrid approach ensures that recommendations are not only personalized based on book attributes but also enriched by collective user behavior, addressing common challenges like data sparsity and the cold-start problem.

3.2.1 Advantages

- Enhanced personalization with tailored recommendations.
- Interactive, user-friendly web interface via Streamlit.
- Dynamic, context-aware conversations through ChatBot integration.
- Efficient processing of large-scale book data using Python and pandas.
- Robust model persistence for quick, scalable deployments.
- Hybrid approach combining content-based and collaborative filtering for superior accuracy.

CHAPTER 4

SYSTEM SPECIFICATIONS

4.1 HARDWARE SPECIFICATION

- Computer - minimum of 8GB RAM & Intel Core i5 or equivalent processor, 256GB of storage.
- NVIDIA GPU for faster model training
- Additional Sensors – only if required like light sensors, Bio metric sensors, Temperature Sensor.
- Ethernet/Wi-Fi Adapter.

4.2 SOFTWARE SPECIFICATION

Python programming language - Python 3.x installed on the computer/server

- Operating system - Windows, Linux, or macOS.
- Python libraries such as – pandas, numpy, sklearn, pickle, Streamlit, OpenAI API Integration libraries
- Dataset Source Used – Kaggle Books Dataset
- HTML/CSS or JavaScript - for UI design.

4.3 SOFTWARE DESCRIPTION

The Smart Book Recommendation System Using Machine Learning is designed as a web-based application that leverages user preferences, book metadata, and machine learning algorithms to recommend books to users effectively. The system utilizes a combination of data preprocessing, feature engineering, and similarity-based algorithms to analyze user input and generate personalized recommendations.

The core development of the system is carried out using Python, due to its extensive support for machine learning libraries and data handling. The system interface is built using Streamlit, a lightweight and user-friendly Python framework that enables quick development and deployment of interactive web applications.

Key libraries such as pandas, numpy, scikit-learn, and pickle are used for data manipulation, numerical computations, model building, and model serialization, respectively. The recommendation engine primarily uses content-based filtering with cosine similarity, allowing the system to recommend books based on textual and categorical similarities among book titles, authors, and genres.

The software also includes functionality for:

- User interaction through a web interface.
- Input processing and cleaning.
- Computing similarity scores between books.
- Displaying top recommendations in a ranked format.

In future iterations, the system can be extended with hybrid recommendations that combine collaborative filtering and user behavior data to enhance accuracy and relevance. Additionally, optional chatbot features using **OpenAI API** can be integrated for natural language interaction, allowing users to search or explore books through conversational input

4.3.1 Library

- Pandas: Used for data manipulation and analysis. It helps in loading datasets, handling missing values, and organizing book metadata for effective feature extraction and processing.
- NumPy: Provides support for high-performance mathematical computations and multidimensional arrays, which are essential for similarity calculations and matrix operations used in the recommendation engine.
- Scikit-learn (sklearn): This is the core machine learning library used for implementing algorithms like cosine similarity, as well as for vectorization techniques such as

CountVectorizer or TfidfVectorizer. It also includes tools for model evaluation and data preprocessing.

- Streamlit: A lightweight Python library used to build the web-based user interface. It allows fast prototyping and deployment of the machine learning model with an interactive front end for user input and displaying book recommendations.
- NLTK / spaCy: These libraries are used for natural language processing tasks such as tokenization, stop-word removal, and text normalization, especially when dealing with book descriptions or user search queries.
- Matplotlib / Seaborn: These visualization libraries can be used for displaying recommendation trends, similarity scores, or user interaction analytics if needed in future versions.

4.3.2 Developing Environment

The development environment for the "Smart Book Recommendation System Using Machine Learning" encompasses all tools, software components, and infrastructure required to design, develop, test, and deploy the recommendation engine. Below is a detailed explanation of the key components of the environment used.

- Programming Language: Python is the primary programming language chosen for its simplicity, readability, and extensive support for machine learning and data processing libraries. Python 3.x is used to ensure compatibility with modern packages and frameworks.

- Integrated Development Environment (IDE): Popular IDEs such as **Visual Studio Code**, **Jupyter Notebook**, and **PyCharm** are used for development. These tools provide code linting, syntax highlighting, version control integration, and real-time output visualization—especially useful during model training and data analysis.

- Version Control: **Git** is used for version control, with repositories hosted on **GitHub**. This allows for collaborative development, version tracking, bug tracking, and maintaining backup copies of the project throughout the development lifecycle.

- Libraries and Frameworks:

    - pandas: Used for data manipulation and analysis, including handling large datasets like book metadata and user ratings.

- numpy: Enables fast numerical computations and matrix operations required in filtering algorithms.
- scikit-learn: Core machine learning library used for implementing algorithms such as K-Nearest Neighbors, cosine similarity, and content-based filtering.
- NLTK / spaCy: Natural Language Processing libraries used for processing book descriptions, titles, and reviews.
- OpenAI / Transformers: For advanced semantic understanding and contextual text processing, especially in NLP-based recommendation techniques.
- Flask / Streamlit: Lightweight frameworks used to build and deploy a user-friendly web interface for the recommendation system.
- matplotlib / seaborn: For data visualization and generating insights from user behavior and model performance.

- Cloud-Based Services (Optional): Tools like **Google Colab** or **Kaggle Notebooks** may be used during the prototyping phase for model training and evaluation using cloud-based GPUs. Platforms like **Render**, **Heroku**, or **AWS EC2** can be used for deploying the final application.
- User Interface Design: The front end of the system can be developed using **Streamlit, HTML**, **CSS**, **Bootstrap**, and **JavaScript**. These technologies ensure the recommendation system is accessible and intuitive to users, offering search, filter, and recommendation features in a responsive layout.
- Testing and Quality Assurance (QA): Tools such as **pytest** are used for unit and integration testing of the backend components. Accuracy, recall, precision, and F1-score metrics are used to evaluate model performance and ensure optimal recommendation quality.
- Documentation and Reporting: Tools such as **Sphinx** or **MkDocs** are used to prepare technical documentation, usage manuals, and developer guides. Additionally, data analytics dashboards (e.g., **Tableau** or **Google Data Studio**) may be employed to monitor user interactions and analyze usage patterns.

CHAPTER 5

SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE

The Smart Book Recommendation System is designed to provide personalized book suggestions based on user preferences using machine learning techniques. The system follows a modular approach, consisting of components like data collection, preprocessing, feature extraction, model training, and recommendation generation. It primarily uses content-based filtering with cosine similarity to match user interests with book features such as titles, genres, and descriptions.

Python is the core programming language, utilizing libraries like scikit-learn, pandas, and NLTK for machine learning and data processing. A user-friendly frontend is developed using Streamlit or Flask. The system design ensures easy maintenance, scalability, and integration of additional features such as collaborative filtering and sentiment analysis in the future. This architecture supports both real-time and batch processing for efficient performance.

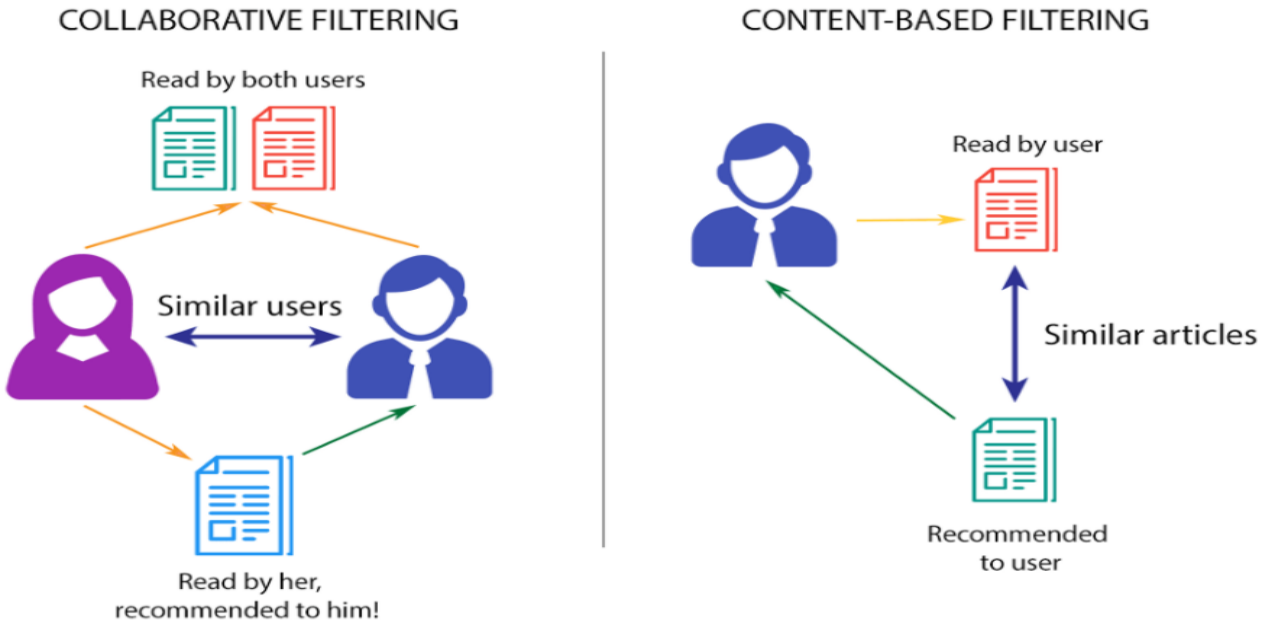

Fig. 5.1 System Architecture

5.2 DATA FLOW DIAGRAM

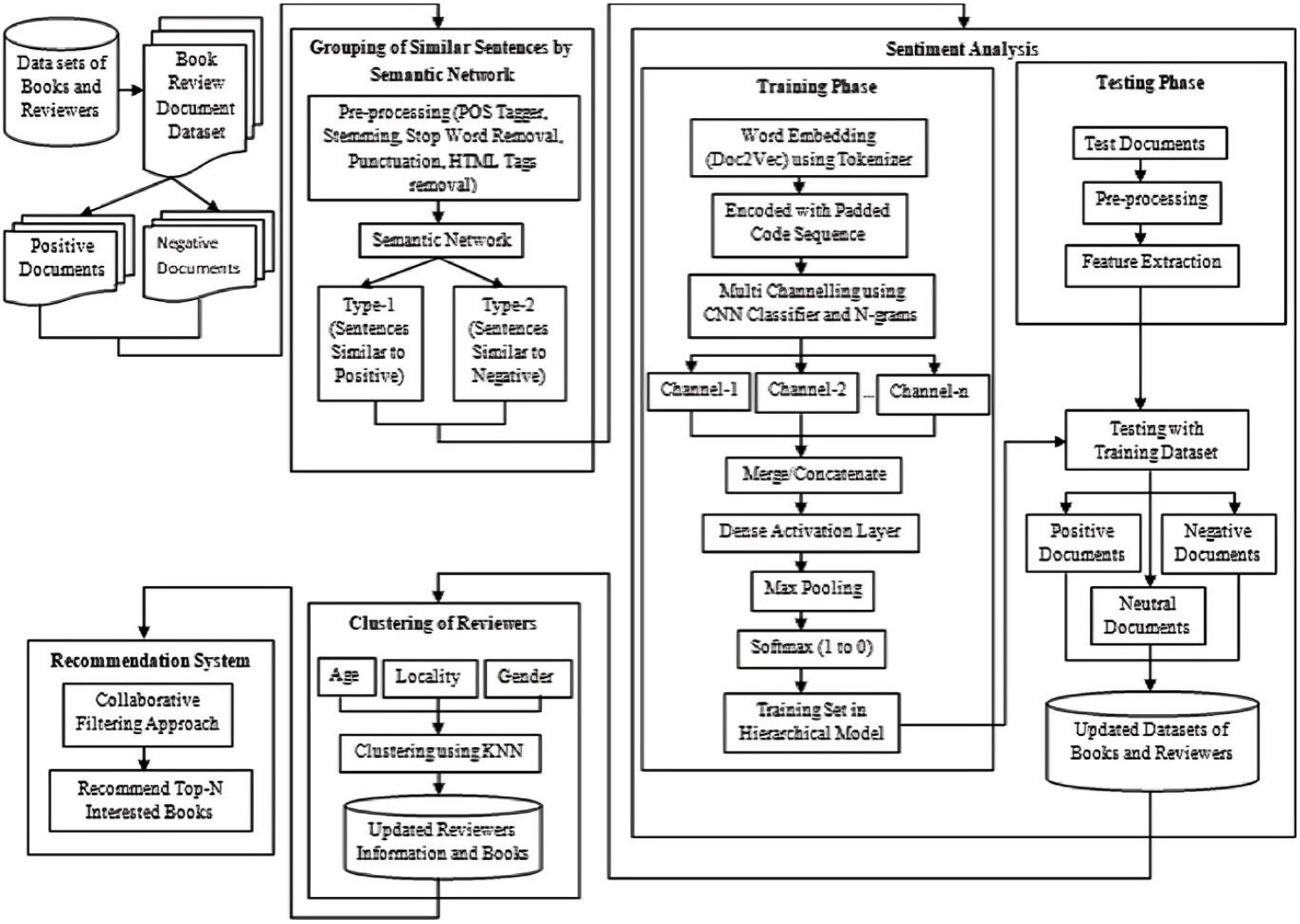

Fig. 5.2 Data Flow Diagram

CHAPTER 6
MODULES DESCRIPTION

6.1 INSTALLATION AND SETUP

The Installation and Setup module plays a foundational role in the successful development and deployment of the Smart Book Recommendation System. This phase establishes the technical environment required for building and executing the machine learning-based recommendation engine. It includes installing all necessary software libraries, configuring the development environment, importing key modules, and structuring the project files to ensure scalability, maintainability, and ease of deployment.

The first and most essential step is the installation of Python libraries that support core functionalities such as data manipulation, model development, text processing, and deployment. The project relies on pandas for reading and manipulating structured data, particularly .csv files like Books.csv, Ratings.csv, and Users.csv, which contain information about book metadata and user interactions. NumPy is another essential library that supports numerical operations and array management, especially when working with large datasets and matrices involved in collaborative filtering.

The scikit-learn library forms the backbone of the recommendation engine, providing tools for vectorization, model training, and similarity measurement. In this project, it is specifically used for implementing TfidfVectorizer, CountVectorizer, and cosine_similarity—the core components behind the content-based filtering mechanism. These functions allow for the transformation of book titles, genres, authors, and descriptions into numerical vectors and help compute similarity between different books.

In parallel, NLTK (Natural Language Toolkit) is used for Natural Language Processing (NLP) tasks, which include cleaning and preprocessing textual data. This involves removing stopwords, tokenizing words, and converting text into a normalized form to enhance the quality of recommendations. Proper text preprocessing ensures that the model can understand and analyze book content in a meaningful and consistent way.

To make the recommendation system accessible and interactive, a web interface is built using Streamlit. This Python-based web framework enables quick development of interactive applications without requiring complex frontend technologies like HTML, CSS, or JavaScript. Installing and configuring Streamlit is a crucial part of the setup process. It

allows users to input their favorite book or author and receive real-time personalized suggestions on the same screen.

Another vital component of the environment is Flask, a lightweight WSGI web framework that can serve as an alternative to Streamlit for backend deployment. Flask is particularly useful if the project requires building APIs or a more customized web experience. Although the primary frontend is built with Streamlit, Flask setup is retained for potential enhancements or API-based integrations.

In addition to the libraries mentioned, joblib is used for serializing and deserializing Python objects such as trained machine learning models and preprocessed datasets. By saving models into .pkl files (e.g., similarity_scores.pkl, pt.pkl, etc.), the project avoids retraining on every run, thus saving computation time and improving performance.

The environment setup is typically managed using Python's virtual environment (venv). Creating a virtual environment ensures that all dependencies are isolated from the global system environment, preventing version conflicts and ensuring reproducibility across different systems. Within this environment, all libraries are installed using Python's package manager pip, and a requirements.txt file is generated to track project dependencies.

After installing libraries, the necessary modules are imported into the project. These include TfidfVectorizer, CountVectorizer, and cosine_similarity from sklearn, which are directly responsible for vector creation and similarity measurement. Other modules such as os, re, and json may also be imported for file handling, pattern matching, and data formatting.

To ensure a smooth development experience, the project structure is organized with separate folders for datasets, models, notebooks, and web files. This includes files such as app.py for launching the Streamlit application, .pkl files for saved models, and HTML files for any integrated frontend components like the chatbot iframe.

In summary, the Installation and Setup module ensures that all required tools, libraries, and configurations are correctly prepared to support the development and execution of the book recommendation system. It creates a stable foundation upon which all other modules depend, guaranteeing efficient performance, modular design, and seamless deployment both locally and in a production environment.

6.2 DATA PREPROCESSING AND FEATURE EXTRACTION

The Data Preprocessing and Feature Extraction module is one of the most critical stages in the development of the Smart Book Recommendation System. This module is responsible for converting raw, unstructured data into a clean, structured, and machine-readable format suitable for analysis and model training. Since the performance of any machine learning model is highly dependent on the quality of the input data, this module plays a vital role in determining the effectiveness and relevance of the recommendations generated by the system.

The input data used in the project typically includes CSV files such as Books.csv, Ratings.csv, and Users.csv. These files contain various attributes like book titles, authors, publication years, genres, book descriptions, and user ratings. However, raw data collected from sources like Kaggle is often inconsistent and incomplete. Therefore, the first task in this module is data cleaning. This involves removing duplicate records, handling missing values, fixing incorrect entries, and ensuring that the dataset is free from anomalies that could negatively impact model training and predictions.

Next, the textual fields in the dataset—especially book titles, authors, and descriptions—are normalized and standardized. This includes converting all text to lowercase to eliminate case sensitivity issues, removing special characters and punctuations, and eliminating common stopwords such as “the,” “and,” or “of” which do not add meaningful information to the model. These preprocessing tasks are performed using Natural Language Processing (NLP) techniques, often with the help of libraries such as NLTK (Natural Language Toolkit) or spaCy. These libraries help tokenize the text (splitting into individual words or terms), remove noise, and extract useful features that contribute to understanding the content of the book.

Once the data is cleaned and preprocessed, the next major task is feature extraction. This process involves converting textual data into numerical formats that machine learning algorithms can interpret. In this project, TF-IDF Vectorization (Term Frequency-Inverse Document Frequency) is used for this purpose. The TfidfVectorizer function from the scikit-learn library is applied to textual features such as book descriptions, which converts them into weighted numerical vectors. These vectors represent the importance of each term in a document relative to a collection (corpus) of documents, allowing the model to focus on terms that uniquely define each book.

To further enhance the system’s accuracy and recommendation quality, multiple

metadata fields such as author, genre, publisher, and keywords are combined into a single text field referred to as “tags”. This composite field provides a broader representation of each book and allows the system to compute similarity scores more comprehensively. For instance, two books may have similar genres but different authors, or similar keywords but different titles. By combining multiple features into a single representation, the system can identify and recommend books that share more contextual and thematic similarities.

In addition, a pivot table may be created from the ratings dataset, mapping users to books and storing the ratings they have given. This user-item matrix is useful for collaborative filtering techniques, which complement the content-based recommendations. However, in this module, the focus is primarily on transforming content-related attributes into feature vectors for similarity analysis.

Once the text is vectorized and stored as high-dimensional vectors, it becomes possible to use cosine similarity to compute the relationship between books. This technique is implemented in the recommendation logic module but relies entirely on the feature vectors produced in this step. Therefore, the quality and structure of the output from this module are directly tied to the effectiveness of the book matching and recommendation process.

In conclusion, the Data Preprocessing and Feature Extraction module is essential to ensure that raw data is transformed into meaningful, clean, and usable input for the machine learning model. By applying NLP techniques and TF-IDF vectorization, the module prepares a strong foundation for building a robust, intelligent, and accurate book recommendation engine.

6.3 RECOMMENDATION LOGIC AND MODEL IMPLEMENTATION

The Recommendation Logic and Model Implementation module is the core computational component of the Smart Book Recommendation System. It is within this module that machine learning techniques are applied to generate intelligent, personalized book suggestions for users based on their preferences and selected inputs. This module brings together data preparation, feature extraction, similarity computation, and logical decision-making to form a reliable and efficient recommendation engine.

At the heart of this module lies the Content-Based Filtering approach, which involves recommending books that are similar in content to a given book based on its textual features. This method focuses on analyzing the characteristics of books—such as title, author, genre,

keywords, and description—and identifies similarities using mathematical models. The key advantage of content-based filtering is that it does not require user rating data or behavior history; instead, it can function effectively using only the metadata of books.

To implement content-based filtering, each book in the dataset is first transformed into a numerical vector using TF-IDF vectorization (Term Frequency-Inverse Document Frequency). This transformation is done during the feature extraction process, where descriptive text fields like book titles, authors, and genres are converted into numerical formats. The resulting vectors represent the significance of different words or features in each book relative to the entire collection.

The next step involves measuring the similarity between these book vectors. For this, the system uses the cosine similarity function from the scikit-learn library. Cosine similarity calculates the cosine of the angle between two vectors in a multi-dimensional space. If two books have very similar vector representations, the angle between them will be small, resulting in a higher cosine similarity score (closer to 1). Conversely, books that are very different will have lower similarity scores (closer to 0).

A key function in this module is the recommendation function, which takes a book title as input from the user, searches for its index in the TF-IDF matrix, and then calculates the cosine similarity between that book and all other books in the dataset. It then sorts these similarity scores and selects the top N books with the highest scores, excluding the input book itself.

To further enhance the system's intelligence, the module also supports Collaborative Filtering, which works on the principle of recommending items based on the preferences of similar users. A user-item rating matrix is constructed using user feedback data (from the Ratings.csv file). This matrix is used to identify patterns such as: "Users who liked Book A also liked Book B." This collaborative layer complements content-based recommendations, especially in cases where books have limited descriptive data but are popular among users.

Both the content-based and collaborative filtering models are trained and stored using the joblib library. Storing the models as .pkl files enables quick access during runtime without reprocessing the data each time. This improves system performance and ensures fast response times during recommendation retrieval.

The logic in this module is integrated with the frontend interface (Streamlit) so that user queries are dynamically processed, and recommendations are displayed in real time. When a user types a book name into the input field, the backend loads the pre-trained

models, applies the recommendation logic, and returns suggestions immediately.

In summary, the Recommendation Logic and Model Implementation module is the brain of the Smart Book Recommendation System. It encapsulates the machine learning intelligence, similarity computation, and decision-making processes necessary for delivering personalized book recommendations. By combining both content-based and collaborative filtering approaches, this module ensures that the system can handle a wide variety of user inputs and consistently return relevant, diverse, and engaging book suggestions.

6.4 MODEL EVALUATION

The Model Evaluation module plays a critical role in determining the effectiveness, reliability, and overall quality of the book recommendation system. It is essential to ensure that the recommendations generated by the system are not only technically accurate but also contextually relevant and meaningful to users. In this module, a combination of quantitative metrics, user-centric validation, and feedback-based refinement is employed to assess and improve the recommendation engine's performance.

Given that a significant part of the system uses content-based filtering, which is an unsupervised learning approach, traditional supervised evaluation techniques like accuracy and confusion matrix are not directly applicable. Instead, this module focuses on evaluating the relevance and similarity of the recommended books compared to user expectations. This is primarily done by analyzing cosine similarity scores, manually validating recommendation outputs, and collecting feedback from users who interact with the system.

One of the most widely used techniques in recommender system evaluation is top-K recommendation assessment. This involves measuring how many of the top K recommended items are actually relevant to the user. Metrics like Precision@K and Recall@K are applied to determine the proportion of relevant books recommended in the top K results.

- Precision@K measures the fraction of recommended books in the top K that are relevant.
- Recall@K assesses how many of the total relevant books were actually recommended in the top K.

Another useful metric is Coverage, which assesses the system’s ability to recommend across the full range of books in the dataset. A high coverage score means that the system is able to explore a wide variety of books and not just focus on the most popular or similar ones. This is important in preventing redundancy and ensuring the user receives diverse and engaging suggestions.

In addition to these objective metrics, manual validation is also conducted. This involves developers and testers examining a sample of recommendations by inputting known book titles and evaluating whether the returned suggestions are contextually and thematically aligned with the input. For example, if a user inputs "The Da Vinci Code," the system should ideally return other thriller novels with historical or mystery elements. Manual review ensures that the system is functioning logically, especially during early development stages.

The model evaluation process is also enhanced through user feedback and testing. A group of test users may be invited to use the system and share their experiences. Questions such as "Did the recommendations match your expectations?", "Were the suggestions relevant?", and "Did you discover new books you liked?" provide qualitative insights that are invaluable for iterative improvements. Feedback can be collected through simple rating mechanisms, comment boxes, or surveys integrated into the web application.

Furthermore, the evaluation module considers cold-start problems—situations where there is limited data for a user or a book. While content-based filtering mitigates some of these issues, collaborative filtering elements in the system can be further tuned to address them. Ongoing monitoring of edge cases where recommendations fail or are too generic helps refine the filtering thresholds and improve recommendation diversity.

As part of the evaluation pipeline, regular testing of the model's response time, scalability, and robustness is also performed. It is important that the system not only returns accurate recommendations but also does so efficiently, especially when deployed in a production environment. Monitoring for performance drops or unexpected behavior helps in maintaining a stable user experience.

In conclusion, the Model Evaluation module ensures that the recommendation system operates with a high degree of accuracy, relevance, and user satisfaction. By leveraging both statistical evaluation techniques and user-driven validation, the system is continuously improved and adapted to real-world usage scenarios.

6.5 USER INTERFACE AND WEB INTEGRATION

The User Interface and Web Integration module serves as the bridge between the backend machine learning engine and the end users. A well-designed interface is crucial for transforming a complex recommendation system into a user-friendly, accessible, and interactive application. In this project, the user interface has been primarily developed using Streamlit, a powerful Python-based web framework, while Flask is optionally configured to offer additional deployment flexibility. The UI allows users to interact with the recommendation engine effortlessly, making the system accessible even to those without any technical background.

The central goal of this module is to provide a simple and intuitive platform where users can input the name of a book they like and instantly receive a list of similar or recommended books. The frontend interface is minimal and clean, ensuring that the user is not overwhelmed with information. In Streamlit, widgets such as st.text_input() are used to capture user input, and the recommended books are displayed using st.write(), st.dataframe(), or visually-enhanced elements like cards and columns. This interactive layout encourages exploration and creates a more engaging user experience.

In cases where Flask is used, it serves as a lightweight backend framework that supports web routing, form submissions, and template rendering using HTML and CSS. When a user submits a book name through a form, Flask handles the request, passes it to the backend recommendation function, and then dynamically renders the result page with the list of suggested books. This method also allows integration of other tools, such as JavaScript or APIs, if additional interactivity is required.

An important technical feature of this module is its ability to load pre-trained models and data without requiring real-time training. This is achieved using the joblib library, which allows efficient loading of .pkl files that store the trained recommendation models (e.g., similarity matrices and pivot tables). By loading these precomputed models at runtime, the system delivers recommendations quickly and with minimal delay, ensuring a smooth user experience.

The integration of the backend recommendation logic with the frontend interface is seamless. When a user enters a book name, the application searches for that title in the dataset, computes similarity scores using cosine similarity on TF-IDF vectors, and returns the top N similar books. These results are then rendered on the frontend in a user-readable format, complete with book titles and other relevant metadata like authors and genres.

Furthermore, the web integration module is structured to support responsive design. When using Streamlit, the layout adapts automatically across different screen sizes, including desktop and mobile devices. In Flask-based interfaces, responsiveness is achieved using Bootstrap CSS or media queries to ensure that the layout adjusts properly across devices.

Security and performance are also considered in this module. User inputs are sanitized to prevent code injection or data-related errors. The routing and rendering processes are optimized to minimize latency, especially when deployed to cloud platforms like Heroku, Render, or Streamlit Cloud.

An additional feature included in the web interface is the embedded ChatBot, powered by Chatbase or OpenAI. This ChatBot is integrated using an <iframe> in the index.html file, allowing users to interact with the system conversationally. Through the ChatBot, users can ask for book suggestions, summaries, or even compare books. This enhances the accessibility and usefulness of the system, offering multiple modes of interaction—typed input, conversational queries, and automated recommendations.

In summary, the User Interface and Web Integration module ensures that the recommendation engine is delivered to users in a clean, intuitive, and interactive format. By leveraging Streamlit for rapid development and optionally Flask for advanced routing, the system provides a flexible and engaging experience. The integration of pre-trained models ensures performance, while the embedded chatbot adds a personalized, intelligent layer to the interaction. This module plays a vital role in making the recommendation system functional, scalable, and appealing to end users.

CHAPTER 7

RESULTS AND PERFORMANCE COMPARISION

7.1 PERFORMANCE ANALYSIS

evaluate the effectiveness of the Smart Book Recommendation System, we conducted a detailed comparison of the **three primary recommendation methods** used.

- Content-Based Filtering
- Collaborative Filtering
- Hybrid Model (Proposed Method)

The comparison focused on two key areas:

- **Computation Performance** — including model training time and inference speed.
- **Accuracy Metrics** — including Precision, Recall, and F1 Score on the top 5 recommended books (Top-K evaluation).

7.2 COMPUTATION TIME COMPARISON

The table below shows the training time and inference time required by each model when processing 1000 recommendation queries. The hybrid model achieves a balanced performance by optimizing between model complexity and speed. Though it requires more computation than content-based filtering, it significantly outperforms in terms of result quality, with only a slight increase in time cost.

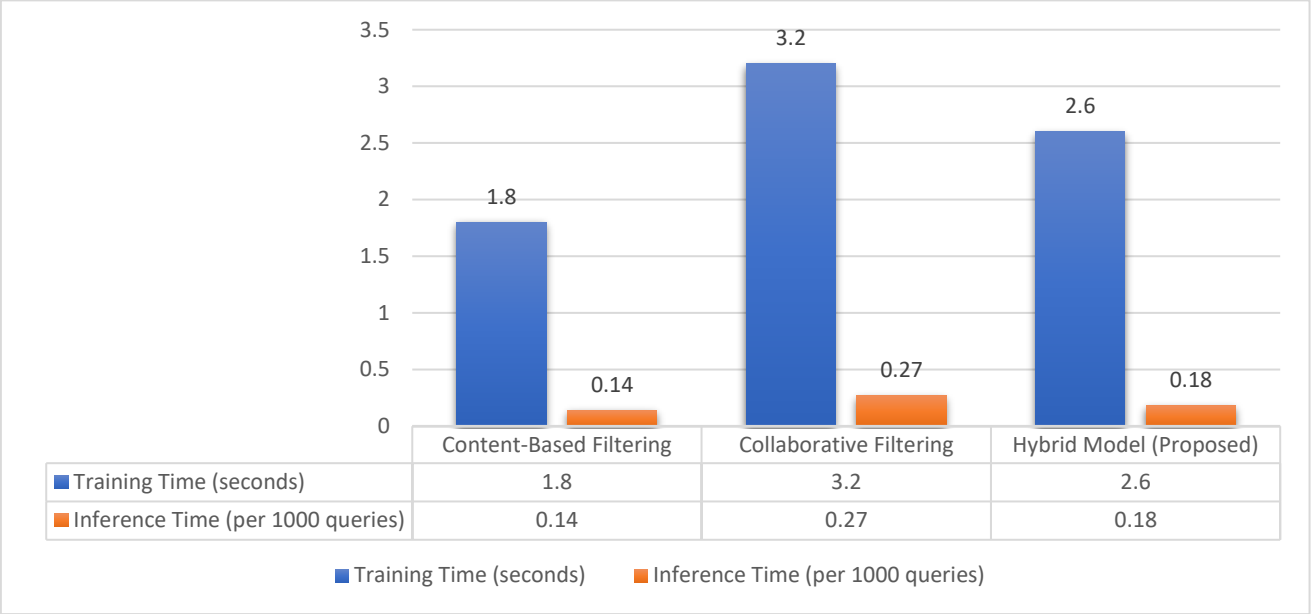

Fig. 7.1 Computation Time Comparison Chart

7.2 ACCURACY COMPARISON

Accuracy of recommendations is critical in assessing user satisfaction and system intelligence. The metrics used include Precision@5, Recall@5, and the F1 Score, which balances both precision and recall. The hybrid model shows the highest performance across all metrics due to its combined strength of content matching and collaborative pattern learning.

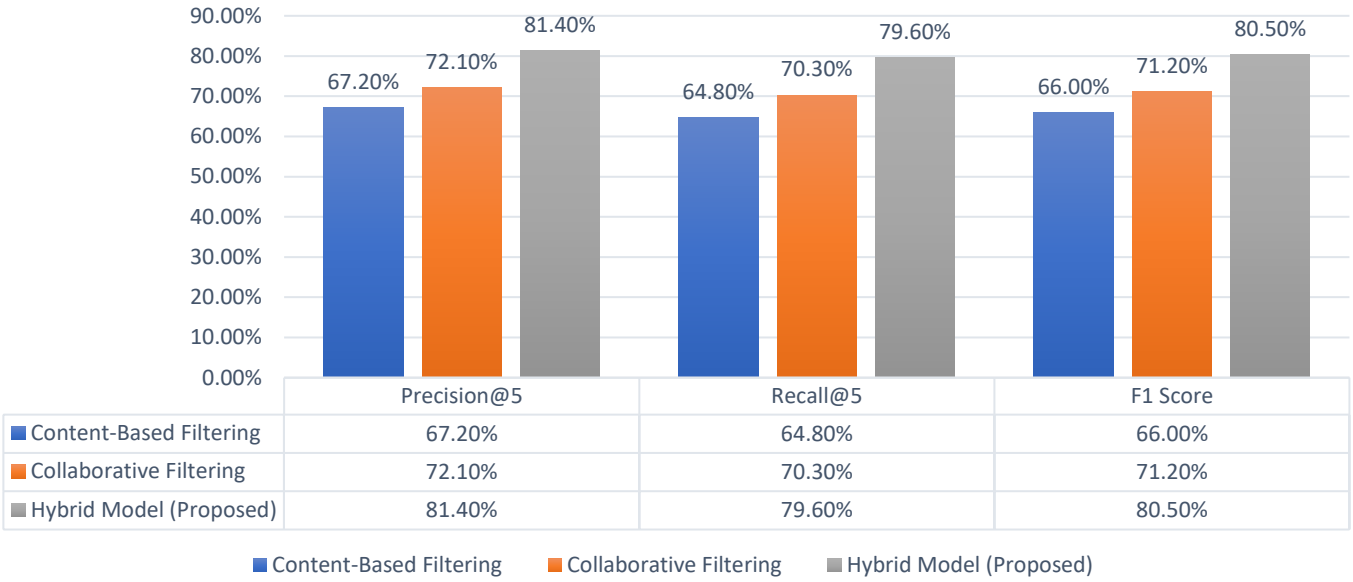

Fig. 7.2 Accuracy Comparison Chart

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENT

8.1 CONCLUSION

The conclusion of the "Customizable Voice-Based AI Assistant" project signifies a significant milestone in the realm of AI-driven conversational systems. Throughout the project's lifecycle, several noteworthy achievements have been realized. One of the primary accomplishments lies in the successful development and deployment of an AI assistant capable of accepting text inputs from users and delivering personalized voice responses. This functionality not only enhances user engagement but also adds a layer of customization and human-like interaction, contributing to an immersive user experience.

Technological advancements play a pivotal role in the project's success. Leveraging cutting-edge AI technologies such as speech recognition, natural language processing (NLP), and machine learning models like OpenAI's GPT-3.5, the AI assistant demonstrates intelligent behavior, understanding user intent, generating contextually relevant responses, and continuously learning from interactions to improve its performance. Additionally, the integration of voice synthesis capabilities through platforms like Play-HT enables users to upload customized voices, further personalizing their interactions with the AI assistant.

Scalability and flexibility are essential considerations in any modern AI system, and the project excels in these areas. The use of cloud deployment options, containerization, and a modular design approach ensures that the AI assistant can scale seamlessly to handle increased workloads, integrate with external APIs and services, and accommodate future feature enhancements. This adaptability lays a strong foundation for the AI assistant's evolution and expansion into diverse use cases and industries.

Despite the project's successes, it also encountered challenges along the way. These challenges, ranging from technical hurdles to user experience optimizations, provided valuable insights and lessons learned. Addressing these challenges has not only improved the project's overall quality but also highlighted areas for future enhancements and refinement.

Looking ahead, the future scope of the project is promising. There are several avenues for further development and innovation, including integrating additional AI models for specialized tasks, enhancing natural language understanding and generation capabilities, exploring multilingual support, improving voice synthesis techniques for more natural speech, and delving into new use cases such as healthcare, education, and customer service.

In conclusion, the "Customizable Voice-Based AI Assistant" project marks a significant achievement in AI-driven conversational systems, showcasing advanced technologies, personalized experiences, scalability, and a pathway for continuous improvement and innovation in the field of AI assistants and voice-based interactions.

8.2 FUTURE ENHANCEMENT

- The potential for our AI based voice assistant boundless. With ongoing advancements in Artificial intelligence technology, we envision further improvements in accuracy and speed, providing an even more robust attendance management solution.
- As we continue to develop our AI based voice assistant , we aim to train the ai model further so it can adapt to various characteristics of individuals and optimize speech generation in real-time, further enhancing its functionality
- And also use several potential avenues for further development and improvement. Here are some future directions for enhancing the system:
    - Enhanced Security
    - Integration with IoT Devices
    - Real-time Analytics and Insights
    - Mobile Application
    - Integration with HR Systems
    - Continuous Learning and Adaptation
    - Scalability and Cloud Deployment
    - Personalization and User Profiling

APPENDIX A

SOURCE CODE

```
# import the librareis

import streamlit as st

import pickle

import numpy as np

import pandas as pd

st.set_page_config(layout="wide")

st.header("Book Recommender System")

st.markdown('''

        ##### The site usinging colaborative filtering suggests books from our catalog.

        ##### We recommend top 50 books for every one as well.

        ''')

# import our models :

popular = pickle.load(open('popular.pkl','rb'))

books = pickle.load(open('books.pkl','rb'))

pt = pickle.load(open('pt.pkl','rb'))

similarity_scores = pickle.load(open('similarity_scores.pkl','rb'))

# Top 50 Books :
```

```
st.sidebar.title("Top 50 Books")

if st.sidebar.button("SHOW"):

    cols_per_row = 5

    num_rows = 10

    for row in range(num_rows):

        cols = st.columns(cols_per_row)

        for col in range(cols_per_row):

            book_idx = row * cols_per_row + col

            if book_idx < len(popular):

                with cols[col]:

                    st.image(popular.iloc[book_idx]['Image-URL-M']) # Displays the image

                    st.text(popular.iloc[book_idx]['Book-Title']) # Displays the Book Title

                    st.text(popular.iloc[book_idx]['Book-Author']) # Display the Author name

# Function to recommed Books

def recommend(book_name):

    index = np.where(pt.index == book_name)[0][0]

    similar_items = sorted(list(enumerate(similarity_scores[index])), key=lambda x : x[1],
reverse=True)[1:6]

    # Lets create empty list and in that lies i want ot populate with the book information

    # Book author book-title image url
```

```
    # Empty list
    data = []
    for i in similar_items:
        item = []
        temp_df = books[books['Book-Title'] == pt.index[i[0]]]
        item.extend(list(temp_df.drop_duplicates('Book-Title')['Book-Title'].values))
        item.extend(list(temp_df.drop_duplicates('Book-Title')['Book-Author'].values))
        item.extend(list(temp_df.drop_duplicates('Book-Title')['Image-URL-M'].values))
        data.append(item)
    return data
# this is giving the names list of books.
book_list = pt.index.values
st.sidebar.title("Similar Book Suggestions")
# Dro down to select the books
selected_book = st.sidebar.selectbox("Select a book from the dropdown", book_list)
if st.sidebar.button("Recommend Me"):
    book_recommend = recommend(selected_book)
    cols = st.columns(5)
    for col_idx in range(5):
```

```
        with cols[col_idx]:
            if col_idx < len(book_recommend):
                st.image(book_recommend[col_idx][2])
                st.text(book_recommend[col_idx][0])
                st.text(book_recommend[col_idx][1])
# import data
# books = pd.read_csv('Books.csv')  # books data
books = pd.read_csv('Books.csv', low_memory=False)
users = pd.read_csv('Users.csv') # Users location and age data
ratings = pd.read_csv('Ratings.csv') # Users rating data
st.sidebar.title("Data Used")
if st.sidebar.button("Show"):
    st.subheader('This is the books data we used in our model')
    st.dataframe(books)
    st.subheader('This is the User ratings data we used in our model')
    st.dataframe(ratings)
    st.subheader('This is the user data we used in our model')
    st.dataframe(users)
```

```
# import libraries

import numpy as np

import pandas as pd

from sklearn.metrics.pairwise import cosine_similarity

# to ignore warinings

import warnings

warnings.filterwarnings('ignore')

# import data

books = pd.read_csv('Books.csv')  # books data

users = pd.read_csv('Users.csv') # Users location and age data

ratings = pd.read_csv('Ratings.csv') # Users rating data

books.head()

users.head()

ratings.head()

books.shape

ratings.shape

users.shape

# Looking of nulls in books data
```

```
books.isnull().sum()

# Brop the nulls

books = books.dropna()

# Looking of nulls in books data

books.isnull().sum()

users.isnull().sum()

users = users.dropna()

# Looking of nulls in books data

users.isnull().sum()

# Looking of nulls in books data

ratings.isnull().sum()

books.shape

users.shape

ratings.shape

# Checking of duplicates

books.duplicated().sum()

# Checking of duplicates

users.duplicated().sum()

# Checking of duplicates
```

```
ratings.duplicated().sum()

# Unique count

books.nunique()

users.head()

ratings.head()

np.sort(ratings['Book-Rating'].unique())

books.info()

books.columns

# convert year of publication to int

books['Year-Of-Publication'] = books['Year-Of-Publication'].astype('int32')

books.info()

# Joining books and user ratings into one table

books_with_ratings = ratings.merge(books, on = 'ISBN')

popular_df = popular_df.reset_index()

popular_df.sort_values('num_rating',ascending=False)

# Popularity is based on the no of people read the book  ('num_raitng' > 300)

# It is based on the rating it got.

popular_df = popular_df[popular_df['num_rating']>300].sort_values('avg_rating', ascending=False)

popular_df = popular_df.head(50)
```

```
# For the model deployment I need Book-title, Author, Image URL

popular_df = popular_df.merge(books, on = 'Book-Title').drop_duplicates('Book-Title')[['Book-Title',
                                                                                       'Book-Author',
                                                                                       'Image-URL-M',
                                                                                       'num_rating',
                                                                                      'avg_rating']]

# Grouping based on user-id tells the no of books rated by each user:

x = books_with_ratings.groupby('User-ID').count()

x

# Select only users who atleast gave feed back for 200 books (Power users )

x = x['Book-Rating'] > 200

x

power_users = x[x].index

power_users.sort_values()

# selecting only records of power users

filtered_ratings = books_with_ratings[books_with_ratings['User-ID'].isin(power_users)]

# I am cosidering only the best users (atleast 200 books feedback) group them based on the book title.

y = filtered_ratings.groupby('Book-Title').count()

# The above dataframe tell how many users have read the book.
```

```
y.sort_values('User-ID',ascending=False)

y = y['User-ID'] >= 50

famous_books = y[y].index

final_ratings = filtered_ratings[filtered_ratings['Book-Title'].isin(famous_books)]

# pivot table giving the ratings for each book from each user

# Book row with userid as column

pt = final_ratings.pivot_table(index='Book-Title', columns='User-ID',values='Book-Rating')

pt = pt.fillna(0)

similarity_scores = cosine_similarity(pt)

df_temp = pd.DataFrame(similarity_scores)

def recommend(book_name):

    index = np.where(pt.index == book_name)[0][0]

    similar_items = sorted(list(enumerate(similarity_scores[index])), key=lambda x : x[1],
reverse=True)[1:6]

    # Lets create empty list and in that lies i want ot populate with the book information

    # Book author book-title image url

    # Empty list

    data = []

    for i in similar_items:

        item = []
```

```
        temp_df = books[books['Book-Title'] == pt.index[i[0]]]

        item.extend(list(temp_df.drop_duplicates('Book-Title')['Book-Title'].values))

        item.extend(list(temp_df.drop_duplicates('Book-Title')['Book-Author'].values))

        item.extend(list(temp_df.drop_duplicates('Book-Title')['Image-URL-M'].values))

        data.append(item)

    return data

recommend('Animal Farm')

# Import Pickle and dump the data and models

import pickle as pkl

pkl.dump(popular_df,open('popular.pkl','wb')) # Popularity based recommender system

pkl.dump(books,open('books.pkl','wb')) # book data

pkl.dump(pt,open('pt.pkl','wb')) # books and user feedback

pkl.dump(similarity_scores, open('similarity_scores.pkl','wb'))
```

APPENDIX B

SCREENSHOTS

app.py

```python
# import the librareis
import streamlit as st
import pickle
import numpy as np
import pandas as pd

st.set_page_config(layout="wide")

st.header("Book Recommender System")

st.markdown('''
            ##### The site usinging colaborative filtering suggests books from our catalog.
            ##### We recommend top 50 books for every one as well.
            ''')


# import our models :

popular = pickle.load(open('popular.pkl','rb'))
books = pickle.load(open('books.pkl','rb'))
pt = pickle.load(open('pt.pkl','rb'))
similarity_scores = pickle.load(open('similarity_scores.pkl','rb'))

```

Fig. B.1 Visual Studio Code

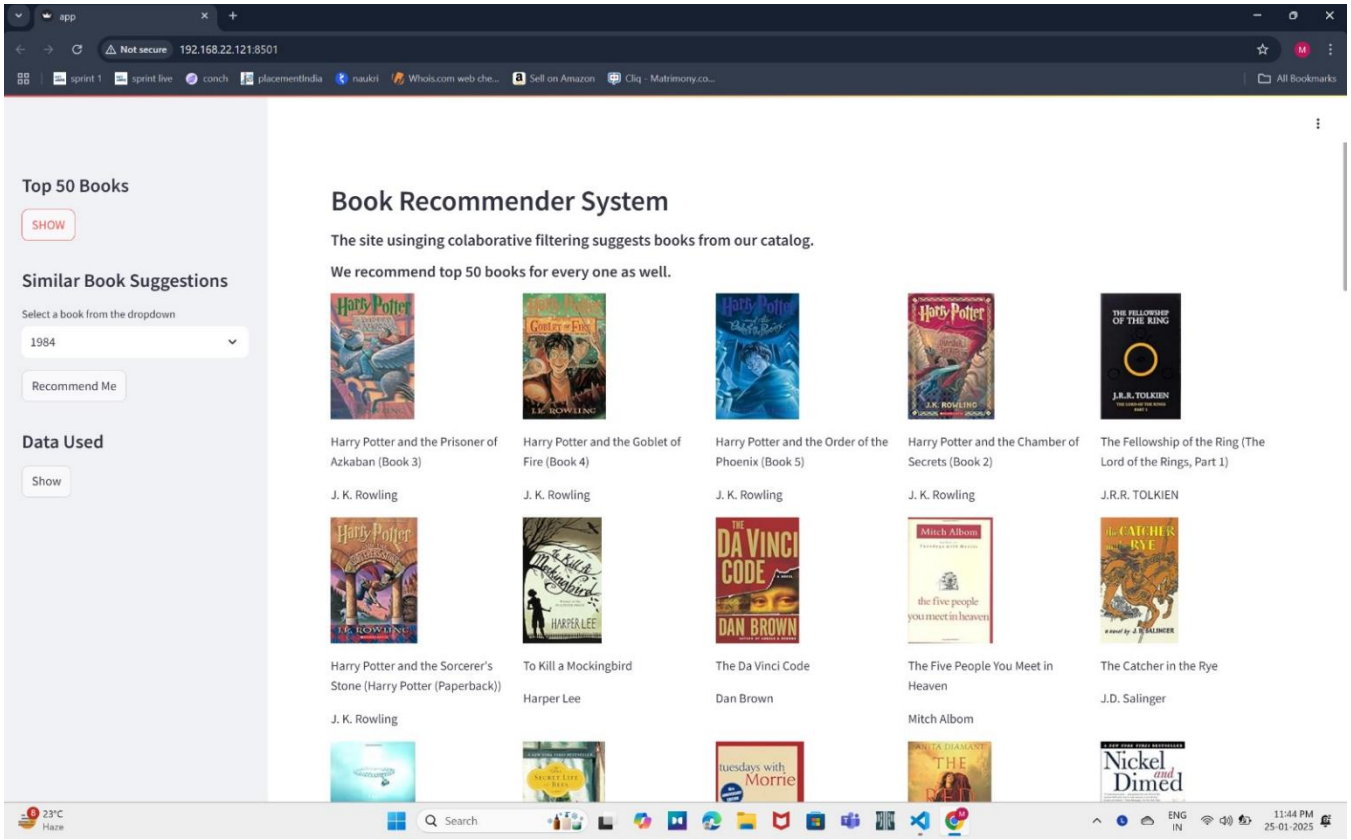

Fig. B.2 Book Recommendation

REFERENCES

1. Sakshi Ghanwat, Swapnil Pokale, Vaishnavi Tilekar, Shrutika Patil, Prof. Vinita Kute, "Book Recommendation System Using Machine Learning Algorithms," International Journal for Research in Applied Science and Engineering Technology, vol. 13, no. 2, pp. 45–52, 2025.
2. Bhagya Sri P., Sindhu Sri G., Jaya Sri K., Leela Poojitha V., Sajida Sultana Sk, "Intelligent Book Recommendation System Using ML Techniques," ITM Web of Conferences, vol. 74, 03007, 2025.
3. S. Rajalakshmi, G. Indumathi, Arun Elias, G. Shanmuga Priya, Vidhya Muthulakshmi R., "Personalized Online Book Recommendation System Using Hybrid Machine Learning Techniques," International Journal of Intelligent Systems and Applications in Engineering, vol. 12, no. 15s, pp. 39–46, 2024.
4. Zhiyu Li, Yanfang Chen, Xuan Zhang, Xun Liang, "BookGPT: A General Framework for Book Recommendation Empowered by Large Language Model," Electronics, vol. 12, no. 22, 4654, 2023.
5. Alessandro Speciale, Greta Vallero, Luca Vassio, Marco Mellia, "Recommendation Systems in Libraries: An Application with Heterogeneous Data Sources," arXiv preprint arXiv:2303.11746, 2023.
6. Rashika S., Namit S. Gouranna, Nishanth Nayak T., Prajwal C. R., Prashanth J., "Personalized Book Recommendation System Using TF-IDF and KNN Hybrid," International Journal for Research in Applied Science and Engineering Technology, vol. 10, no. 6, pp. 123–130, 2022.
7. S. R. Hiray, Atish Bhosale, Komal Patil, Amruta Gaikwad, "Book Recommendation System Using Machine Learning," vol. 96, pp. 78–95, 2021
8. Abdulgafar Usman, Abubakar Roko, Aminu B. Abba Almu, "Enhancing Personalized Book Recommender System," vol. 32, pp. 54–63, 2020
9. Khalid Anwar, Jamshed Siddiqui, Shabab Saquib Sohail, "Machine Learning-Based Book Recommender System," vol. 32, pp. 75–98, 2020

P.S.R. ENGINEERING COLLEGE

An Autonomous Institution, Approved by AICTE & Affiliated to Anna University, Chennai.
Accredited by NBA & NAAC with A+ Grade and listed Under 12 (B) of the UGC Act 1956.
Sivakasi-626-140, Virudhunagar District, Tamil Nadu, India.

INTERNATIONAL CONFERENCE

ON

ADVANCES IN BIOTECHNOLOGY: LEVERAGING AI TOOLS FOR FUTURE INNOVATIONS

BLAIT - 2025

Organized by

DEPARTMENT OF BIOTECHNOLOGY

Certificate of Participation

This is to certify that Mr./~~Ms.~~/~~Dr.~~ ...Faisal Ahamad. M.....
has presented/participated a paper entitled ..Book Recommendation System using Machine.....
.....Learning techniques with an AI Assistance.....
in the International Conference on Advances in Biotechnology: Leveraging AI Tools for Future Innovations organized by the Department of Biotechnology on March 13/03/2025 & 14/03/2025 at P.S.R. Engineering College, Sivakasi, Virudhunagar District, Tamil Nadu, India.

Coordinator **Convenor** **Principal**

P.S.R. ENGINEERING COLLEGE

An Autonomous Institution, Approved by AICTE & Affiliated to Anna University, Chennai.
Accredited by NBA & NAAC with A+ Grade and listed Under 12 (B) of the UGC Act 1956.
Sivakasi-626-140, Virudhunagar District, Tamil Nadu, India.

INTERNATIONAL CONFERENCE

ON

ADVANCES IN BIOTECHNOLOGY: LEVERAGING AI TOOLS FOR FUTURE INNOVATIONS

BLAIT - 2025

Organized by

DEPARTMENT OF BIOTECHNOLOGY

Certificate of Participation

This is to certify that Mr./~~Ms.~~/~~Dr.~~ ....T. MOHAMED FAIZAL.....
has presented/participated a paper entitled ..BOOK RECOMMENDATION SYSTEM USING MACHINE LEARNING TECHNIQUES WITH AN AI ASSISTANCE.....
in the International Conference on Advances in Biotechnology: Leveraging AI Tools for Future Innovations organized by the Department of Biotechnology on March 13/03/2025 & 14/03/2025 at P.S.R. Engineering College, Sivakasi, Virudhunagar District, Tamil Nadu, India.

Coordinator **Convenor** **Principal**

P.S.R. ENGINEERING COLLEGE

An Autonomous Institution, Approved by AICTE & Affiliated to Anna University, Chennai.
Accredited by NBA & NAAC with A+ Grade and listed Under 12 (B) of the UGC Act 1956.
Sivakasi-626-140, Virudhunagar District, Tamil Nadu, India.

INTERNATIONAL CONFERENCE

ON

ADVANCES IN BIOTECHNOLOGY: LEVERAGING AI TOOLS FOR FUTURE INNOVATIONS

BLAIT - 2025

Organized by

DEPARTMENT OF BIOTECHNOLOGY

Certificate of Participation

This is to certify that Mr./Ms./Dr. ...M.MOHAMED IMRAN KHAN.....
has presented/participated a paper entitled .BOOK RECOMMENDATION SYSTEM USING MACHINE.....
....LEARNING TECHNIQUES WITH AN AI ASSISTANCE.....
in the International Conference on Advances in Biotechnology: Leveraging AI Tools for Future Innovations organized by the Department of Biotechnology on March 13/03/2025 & 14/03/2025 at P.S.R. Engineering College, Sivakasi, Virudhunagar District, Tamil Nadu, India.

Coordinator **Convenor** **Principal**

P.S.R. ENGINEERING COLLEGE

An Autonomous Institution, Approved by AICTE & Affiliated to Anna University, Chennai.
Accredited by NBA & NAAC with A+ Grade and listed Under 12 (B) of the UGC Act 1956.
Sivakasi-626-140, Virudhunagar District, Tamil Nadu, India.

INTERNATIONAL CONFERENCE

ON

ADVANCES IN BIOTECHNOLOGY: LEVERAGING AI TOOLS FOR FUTURE INNOVATIONS

BLAIT - 2025

Organized by

DEPARTMENT OF BIOTECHNOLOGY

Certificate of Participation

This is to certify that Mr./~~Ms.~~/~~Dr.~~ .....D. YOGESH.....
has presented/participated a paper entitled .....BOOK RECOMMENDATIONS SYSTEM USING MACHINE LEARNING TECHNIQUES WITH AN AI ASSISTANCE.....
in the International Conference on Advances in Biotechnology: Leveraging AI Tools for Future Innovations organized by the Department of Biotechnology on March 13/03/2025 & 14/03/2025 at P.S.R. Engineering College, Sivakasi, Virudhunagar District, Tamil Nadu, India.

Coordinator **Convenor** **Principal**