Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavre

A Project Report
on
"NepLearn: Smart Learning for Every Student"

[Code No.: COMP 206]

(For partial fulfillment of Year II / Semester I in Computer Engineering)

Submitted by
Prabin Babu Basel (Regd. No. 037958-24)
Saksham Dallakoti (Regd. No. 037965-24)
Sahaj Wagle (Regd. No. 038012-24)

Submitted to
Mr. Suman Shrestha
Department of Computer Science and Engineering

February 9, 2026

Bona fide Certificate

This project work on

“NepLearn: Smart Learning for Every Student”

is the bona fide work of

“

Prabin Babu Basel (Regd. No. 037958-24),

Saksham Dallakoti (Regd. No. 037965-24),

Sahaj Wagle (Regd. No. 038012-24)

”

who carried out the project work under my supervision.

Project Supervisor

Ms. Subhadra Joshi
Lecturer
Department of Computer Science and Engineering

Acknowledgements

We would like to express our sincere gratitude to everyone who contributed to the successful completion of this project, "NepLearn: Smart Learning for Every Student."

First and foremost, we would like to extend our deepest thanks to our project supervisor, Ms. Subhadra Joshi, from the Department of Computer Science and Engineering, Kathmandu University. Her continuous guidance, technical support, and valuable feedback were extremely important throughout the project. In particular, her support in areas such as machine learning concepts, data preprocessing, clustering,feature extraction, model training, and performance improvement helped us greatly in building and understanding the question recommendation system. Her encouragement and problem-solving approach guided us at every stage of development.

We are also thankful to the Department of Computer Science and Engineering, Kathmandu University, for providing the necessary resources, computing facilities, and a positive learning environment that made it possible to complete this project successfully.

We would like to express our appreciation to our team members: Sahaj Wagle, Saksham Dallakoti, Prabin Babu Basel, and for their teamwork, dedication, and willingness to learn. The collaborative effort, shared responsibility, and mutual support among team members played a key role in overcoming challenges during the project.

Finally, we would like to thank our families and friends for their constant encouragement, patience, and motivation throughout the duration of this project.

Abstract

Preparing effectively for academic examinations has remained a major challenge for students due to the lack of structured, personalized and easily accessible practice resources. Students often face difficulty in identifying important topics, suitable questions, and appropriate difficulty levels for efficient preparation. This project was carried out to design and develop an Artificial Intelligence (AI)-driven Question Recommendation System that centralizes academic questions and provides intelligent, personalized study support.The system clustered more than 1000 questions extracted from past C programming question papers and textbooks using K-means clustering on embeddings generated by a mini language model. Relevant features were extracted from these clusters and used to train Extreme Gradient Boosting (XGBoost) and Random Forest models for predicting question patterns and generating recommendations. The platform was implemented using Python, FastAPI for the backend, and React.js for the frontend, incorporating a secure login system and a chatbot-based conversational interface.The developed system successfully generated topic-based and difficulty-aware question recommendations supporting more targeted and organized study sessions. Overall, the project demonstrated that Machine Learning (ML)-based recommendation techniques can enhance exam preparation by making learning more adaptive and student-centered. Future work may include expanding the dataset, improving model performance, and integrating additional subjects.

Keywords: *Machine Learning, Question Recommendation System, K-means Clustering, XGBoost, Random Forest, Exam Preparation*

Contents

List of Figures

List of Tables

Abbreviations

AI Artificial Intelligence.

API Application Programming Interface.

BERT Bidirectional Encoder Representations from Transformers.

CPU Central Processing Unit.

CUDA Compute Unified Device Architecture.

DBSCAN Density-Based Spatial Clustering of Applications with Noise.

GPU Graphics Processing Unit.

JSON Javascript Object Notation.

LLM Large Language Model.

ML Machine Learning.

MPS Metal Performance Shaders.

NLP Natural Language Processing.

OCR Optical Character Recognition.

PDF Portable Document Format.

RAM Random Access Memory.

RESTful API Representational State Transfer Application Programming Interface.

UI User Interface.

XGBoost Extreme Gradient Boosting.

Chapter 1

Introduction

1.1 Background

In recent years, the use of educational technology has increased significantly as institutions seek to improve learning support and exam preparation. Advances in artificial intelligence and machine learning have enabled the development of intelligent tutoring systems, adaptive learning platforms, and recommendation-based study assistants that provide personalized learning experiences. These systems analyze student data to understand learning patterns and suggest appropriate study materials, making digital platforms an essential part of modern education.

Despite these developments, many existing learning platforms still suffer from limited personalization and poor organization of academic content. Most systems provide generic resources, making it difficult for students to identify important topics, suitable practice questions, and appropriate difficulty levels. In technical subjects such as programming, this limitation often results in inefficient study practices and increased academic stress among undergraduate students.

In the context of Nepal, although online learning resources are gradually expanding, a large number of students continue to struggle with accessing structured and relevant study materials. There is a clear need for a lightweight, web-based system that organizes academic questions and provides intelligent recommendations based on student needs. This project was developed to address these challenges by designing an AI-driven Question Recommendation System that supports personalized question suggestions, targeted practice, and improved exam preparation for undergraduate students.

1.2 Objectives

The main objectives achieved by the project are:

- To develop a machine learning–based system for clustering academic questions and predicting question patterns.
- To provide personalized question recommendations based on extracted features and learning needs of undergraduate students.
- To design and implement a secure web-based platform with login and chatbot-based interaction.
- To integrate a language model for generating answers and supporting conceptual understanding.

1.3 Motivation and Significance

The motivation behind selecting the AI-driven Question Recommendation System as a project stems from the increasing difficulty students face in finding clear, structured, and relevant study materials for effective exam preparation. Although numerous online resources are available, much of the content remains scattered, unorganized, and poorly aligned with individual academic needs. This often leads to confusion, inefficient study habits, and increased anxiety among undergraduate students.

This project was undertaken to bridge the gap between available study resources and the actual needs of students. Unlike traditional preparation methods, the proposed system provides a centralized and intelligent platform that delivers personalized question recommendations based on topics and learning requirements. The system emphasizes simplicity and usability, ensuring that students can easily access focused practice questions without spending excessive time searching through multiple sources.

The implemented system combines features such as question clustering, predictive modeling, chatbot-based interaction, and automated answer generation, making it a complete solution for modern exam preparation. Its significance lies in improving learning efficiency, supporting targeted practice, and enhancing student engagement. Furthermore, the project demonstrates the practical application of machine learning techniques in education and establishes a foundation for future extensions to additional subjects and advanced personalization features.

Chapter 2

Related Works

2.1 Review of Existing Works

The field of educational technology and AI-driven learning systems has seen significant development in recent years. Several automated systems have been developed to support students in exam preparation and personalized learning. NepLearn aims to provide similar services with improved features such as machine learning-based clustering, intelligent pattern recognition from historical exam data, and conversational AI interfaces.

2.1.1 Eklavvya Exam Management Platform

Eklavvya Exam Management Platform Dharmadhikari and Kamat (2023) provides structured question repositories with fields for difficulty, topic, marks, and cognitive level. The system supports analytics for item frequency and difficulty distribution, showing best practices for designing scalable question banks.

Strengths:

- Well-structured question organization with metadata.
- Basic analytics for difficulty and frequency distribution.

Limitations:

- Does not integrate machine-learning recommendation engines.
- No predictive trend modeling from historical exams.

Comparison with NepLearn: NepLearn extends beyond simple metadata tagging by implementing intelligent clustering and predictive models that learn from historical patterns to provide personalized recommendations.

2.1.2 NLP-Based Educational Chatbots

Virtual Teaching Assistants Boateng et al. (2022) using Bidirectional Encoder Representations from Transformers (BERT)-based systems show improvements in intent recognition, question answering, and subject-specific response accuracy. These assistants help students access course content quickly, reducing the instructor's workload.

Strengths:

- Natural conversational interface for student support.
- Effective intent recognition and response generation.

Limitations:

- Do not combine historical exam pattern analysis.

Comparison with NepLearn: NepLearn integrates conversational AI with intelligent question recommendations, combining the benefits of chatbot interfaces with ML-powered exam preparation support.

2.2 Summary

Considering the features of previous projects, NepLearn focuses on combining intelligent question clustering using K-means, predictive modeling with XGBoost and Random Forest, and conversational AI interfaces to create a comprehensive exam preparation platform. This integrated approach enhances efficiency, personalization, and accessibility compared to existing platforms that typically focus on only one or two of these aspects.

Chapter 3

Design and Implementation

3.1 System Overview

The primary objective of the AI-driven Question Recommendation System is to provide a web-based platform that enables undergraduate students to access personalized questions and obtain answers efficiently, while supporting targeted exam preparation. The system bridges the gap between scattered academic resources and structured study guidance through machine learning–based recommendation and language model–generated answers.

The major components of the system include:

- **User Interface (Frontend):** Developed using React.js, it includes login/signup pages, a homepage to select between question generation or answer generation, and a chatbot interface that interacts with users by providing recommended questions or answers.
- **Backend Server:** Built with FastAPI, it handles authentication, processes user requests, and routes inputs to either the question recommendation model or the language model for answers.
- **Machine Learning Module:** Implements K-means/Density-Based Spatial Clustering of Applications with Noise (DBSCAN) clustering for question organization and XGBoost/Random Forest models for pattern prediction and recommendations.
- **Language Model Module:** The system integrates the TinyLlama-1.1B-Chat-v1.0 model obtained from the Hugging Face model hub using the Transformers library.
- **Database Layer:** PostgreSQL database stores user data and user-related information (currently limited to basic storage requirements).

These components interact through secure Application Programming Interface (API) calls. Users log in, navigate the homepage to select question or answer generation, and interact with the chatbot to receive either recommended questions or generated answers. The backend processes user inputs, invokes the appropriate model, and returns results via the chatbot interface, providing a seamless and interactive study experience. A system architecture diagram is included to illustrate the component interactions.

3.2 System Requirement Specifications

This section outlines the software and hardware requirements necessary for the development and deployment of NepLearn.

3.2.1 Software Requirements

The following software tools and technologies were used:

- **Programming languages:** Python, JavaScript
- **Frameworks and libraries:** React.js, FastAPI, scikit-learn, XGBoost,NumPy, pandas, miniLM
- **OCR Tool:** pytesseract for text extraction from Portable Document Format (PDF)s and images
- **Development tools:** Visual Studio Code, Jupyter Notebook, Git, GitHub
- **Database:** PostgreSQL
- **Operating system:** Windows, Linux, Mac-OS

Functional Requirements

The core functional requirements of the system include:

- User registration and authentication.
- Selection between question generation and answer generation modes.
- Input of prompts for requesting questions.
- Generation of answers to user-submitted questions using a language model.
- Chatbot based interaction for displaying questions and answers.

Non-Functional Requirements

The non-functional requirements include:

- **Usability:** Simple web-based user interface.
- **Performance:** Reasonable response time for question recommendation and answer generation.
- **Scalability:** Supports an increasing number of users and query requests.
- **Reliability:** Consistent system availability and correct functioning of modules.
- **Security:** Secure authentication and protection of user credentials.

3.2.2 Hardware Requirements

The NepLearn System does not require specialized hardware components, as it is a software and machine learning–based project executed on standard computing devices. The hardware requirements are limited to general-purpose computers capable of running the application and supporting internet connectivity.

User-side requirements:

- Desktop or laptop computer
- Minimum 4 GB Random Access Memory (RAM)
- Modern web browser (Chrome, Firefox, or Edge)
- Internet connectivity

Development and testing requirements:

- Desktop or laptop computer
- Minimum 8 GB RAM (16 GB recommended for language model integration)
- Sufficient storage for datasets and models
- Internet connectivity

Since the system is executed locally and does not require Graphics Processing Unit (GPU) acceleration, Central Processing Unit (CPU)-based machines are sufficient for development and demonstration purposes.

3.3 System Design

3.3.1 Architectural Design

The system follows a client-server architecture with a clear separation between frontend, backend, and machine learning components. The frontend, implemented in React.js, acts as the client, while the FastAPI backend handles request processing, user authentication, and communication with the PostgreSQL database for storing user login information.

All interactions between the frontend and backend are performed using Representational State Transfer Application Programming Interface (RESTful API)s, which handle user queries and deliver responses from the machine learning models. The machine learning component operates as a service that processes historical question data, performs clustering, and predicts relevant questions using Random Forest and XGBoost classification models.

This modular architecture ensures maintainability, allows independent updates to each component, and supports efficient handling of increasing user requests while providing near real-time responses through the chatbot interface powered by the TinyLlama language model.

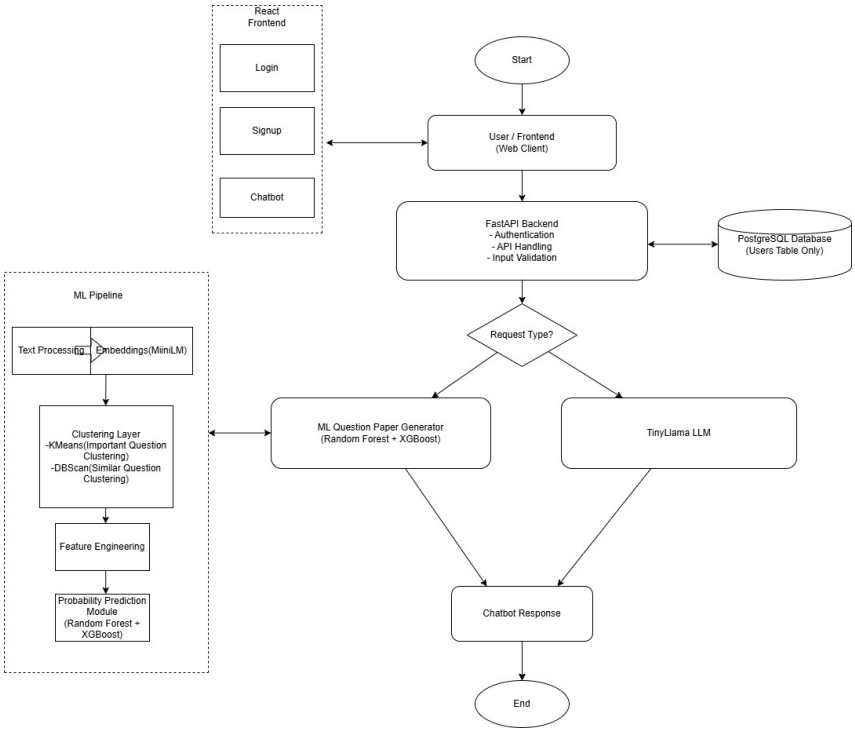

Figure 3.1: System flow

3.3.2 Module or Component Design

The major modules of the system are:

- **Authentication Module:** Manages user registration, login, and session management with secure JSON token-based authentication.
- **Question Processing Module:** Handles question extraction from PDFs using OCR, text preprocessing.
- **Clustering Module:** Implements K-means and DBSCAN clustering on question embeddings to organize questions into semantic groups.
- **Recommendation Module:** Uses XGBoost and Random Forest models to predict and recommend relevant questions based on patterns.
- **Chatbot Interface Module:** Provides conversational AI capabilities using TinyLlama-based Large Language Model (LLM) integration for interactive query handling.
- **User Interface Module:** React.js-based frontend for user interaction, displaying recommended questions, and communicating with the chatbot.

Each module operates independently while communicating through well-defined RESTful APIs implemented using FastAPI., ensuring maintainability and the ability to extend functionality without affecting other components.

3.3.3 Data Design

The dataset is designed to store historical exam questions along with metadata and derived features for machine learning-based question recommendation.

Dataset Structure

It is structured in Javascript Object Notation (JSON) format, with a hierarchical organization: each subject contains a list of questions, and each question contains multiple attributes including embeddings and derived features.

```
{
  "subject": "",
  "questions": [
    {
      "id": "",
      "source": "",
      "subject": "",
      "raw_text": "",
      "cleaned_text": "",
      "embedding": [],
      "topic_id": null,
      "concept_id": null,
      "exam_meta": {
        "year": null,
        "year_semester": "",
        "section": "",
        "marks": null
      },
      "derived_features": {
        "last_seen_year": null,
        "topic_importance": null,
        "concept_importance": null,
        "section_prob": {
          "A": null,
          "B": null,
          "C": null
        },
        "concept_last_seen_year": null,
        "section_confidence": null
      }
    }
  ]
}
```

Figure 3.2: Data structure for each question in the dataset

3.4 Implementation Details

3.4.1 Software Implementation

The frontend of the system was developed using React.js to ensure a responsive and interactive user experience. The interface includes components for user authentication, question display, chatbot interaction, and recommendation visualization.

The backend was implemented using the FastAPI framework, which provides high-performance RESTful APIs with automatic documentation. FastAPI's asynchronous capabilities allow efficient handling of concurrent requests and seamless integration with the machine learning modules.

The database layer uses PostgreSQL to store user authentication and login information securely. API endpoints facilitate communication between the frontend, backend services, and machine learning components using JSON-based data exchange.

3.4.2 Machine Learning Implementation

Dataset Description

The dataset consists of more than 1000 questions extracted from past C programming question papers and textbooks. Questions were collected from various sources including university exam archives and educational materials. Each question was preprocessed to extract relevant text, remove formatting artifacts, and normalize the content.

Data preprocessing steps included text cleaning (removing special characters, normalizing whitespace), tokenization, and embedding generation using a MiniLM language model from Hugging Face (sentence-transformers/all-MiniLM-L6-v2). Each question was converted into a 384-dimensional dense vector, capturing its semantic meaning. These embeddings allow similarity-based clustering using K-Means and DBSCAN, as well as efficient retrieval for generating relevant question recommendations.

Clustering Model:

K-Means clustering was applied to the question embeddings to organize questions into topic-based semantic groups. The number of clusters was selected empirically based on experimental observations. This clustering enables efficient grouping of questions by topic and helps identify common patterns across different subjects, supporting topic-wise analysis and recommendation. In addition, DBSCAN clustering was used to group semantically similar questions based on embedding density, regardless of their topic labels. DBSCAN helps identify closely related questions as well as outliers, allowing the system to detect similar questions that may appear across different exams or sources. Together, these two clustering approaches improve the organization of questions and enhance the relevance of the recommendations generated by the system.

Classification Models:

XGBoost and Random Forest models were used to learn patterns from historical question data and clustering results in order to generate recommendations. Features were derived from the processed question data, including cluster membership and other similarity-based indicators. These models estimate the relevance of questions based on learned patterns and help prioritize questions that are more likely to be useful for student practice and exam preparation.

Training and Evaluation

The classification models were trained using supervised learning with an 80–20 train–validation split. The dataset was divided into training and validation sets to assess how well the models perform on unseen data. The clustering models were applied to the question embeddings to organize questions into meaningful groups, and the clustering parameters were selected empirically based on experimental observations. For the Random Forest and XGBoost models, suitable parameter values were chosen through iterative testing to obtain stable performance. The trained models were then evaluated on the validation set to verify that the system produces consistent and relevant question recommendations for students.

3.5 Algorithms and Flowcharts

This section describes the key algorithms and workflows that govern the operation of the NepLearn system.

3.5.1 Question Processing and Clustering Algorithm

The Question Processing and Clustering algorithm organizes extracted examination questions into meaningful groups based on semantic similarity. Questions are first obtained from PDF files using Optical Character Recognition (OCR) and then cleaned and normalized through preprocessing. Each question is converted into an embedding using a mini language model, after which clustering techniques such as K-Means and DBSCAN group similar questions together. The clustered data is stored and later used to support model training and efficient question retrieval.

3.5.2 Recommendation Generation Algorithm

The Recommendation Generation algorithm creates customized question paper sets according to user-defined parameters such as subject, year, and number of sets. The system validates user input and then uses the QuestionPaperGenerator model to generate papers following predefined structures and rules. Similarity and clustering checks are applied internally to avoid duplicate or closely related questions. All generated sets are compiled into a single structured response and returned to the user.

3.5.3 Summary of Required Content

Table 3.1: Summary of Design and Implementation Content

Section / Component	**Software Project**	**Machine Learning Project**
System Overview	Web-based application modules and workflow (React.js + FastAPI)	Data pipeline-clustering, recommendation, and LLM-based interaction flow
Software Requirements	Python, JavaScript, React.js, FastAPI	scikit-learn, XGBoost, Transformers, NumPy, Pandas, TinyLlama model
System Architecture	Client–server architecture	Training–inference pipeline including clustering, classification, and LLM inference
Module / Component Design	Authentication, UI, API handling modules	Preprocessing, clustering, recommendation, and TinyLlama-based chatbot modules
Database / Data Design	PostgreSQL schema	Dataset structure, embeddings, clustering features
Implementation Details	Code structure, API integration, frontend–backend interaction	Model implementation (K-Means, DBSCAN, Random Forest, XGBoost) and TinyLlama integration
Algorithms and Flowcharts	Application process flows and logic	Clustering, recommendation, and LLM inference algorithms
Evaluation / Testing	Functional and performance testing	Model validation results

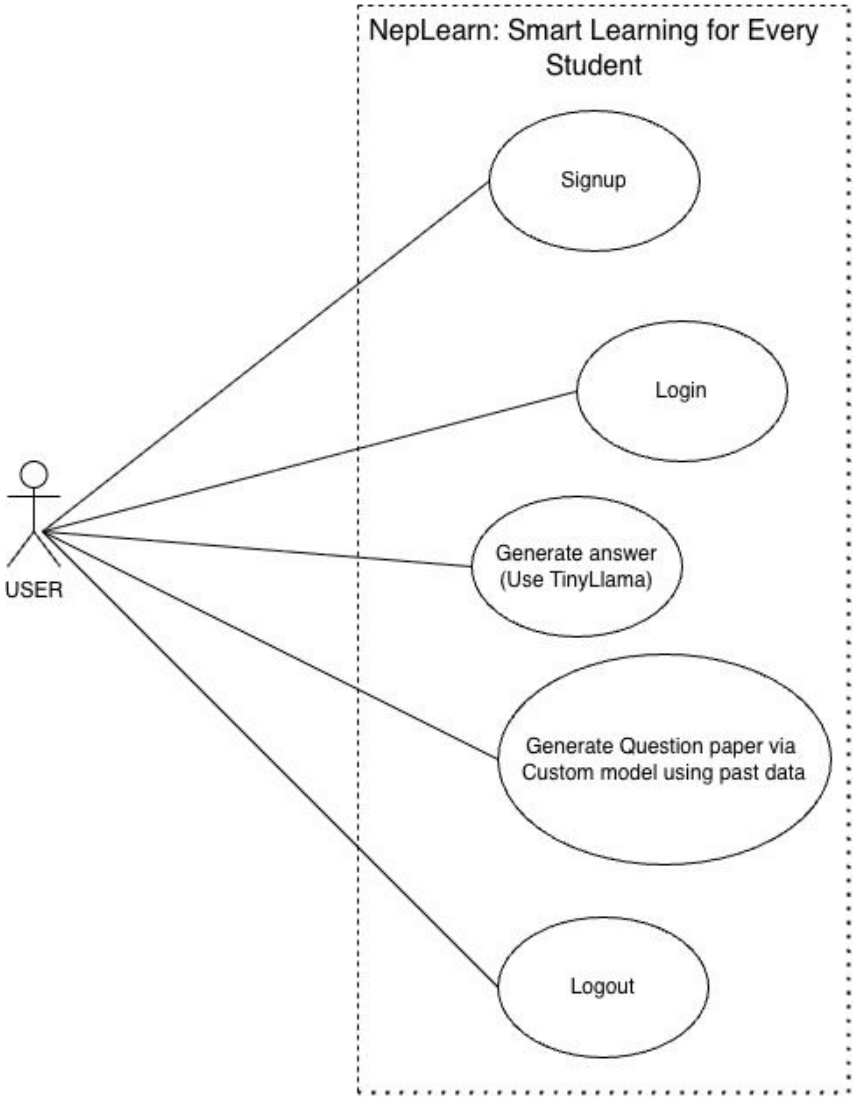

Figure 3.3: Use-case diagram illustrating interactions between users and system functionalities

Chapter 4

Results and Discussion

4.1 Implemented Features

The following key features were successfully implemented:

- **User Authentication and Account Management:** Secure login and signup functionality was implemented to allow users to access the system.
- **Web-Based User Interface:** A React.js based frontend provides a homepage for selecting between question generation and answer generation, along with a chatbot interface for interactive communication.
- **Machine Learning Based Question Recommendation:** The system integrates miniLM embeddings, K-means clustering, feature extraction, and XGBoost and Random Forest models to generate recommended academic questions based on provided prompts.
- **Prompt Based Question Generation:** Users can specify subject, year and number of question sets to receive customized question recommendations.
- **Language Model Based Answer Generation:** A local language model (TinyLlama) is integrated to generate answers to user-submitted questions, supporting conceptual understanding.
- **Backend Services:** FastAPI based APIs handle authentication, process user requests and route inputs to the appropriate machine learning model or language model.

4.2 Results and Performance Analysis

The implemented system functioned correctly across all major components defined in the project. User authentication operations, including signup and login, were successfully handled by the FastAPI backend using RESTful APIs. User credentials were securely stored in the relational database with password hashing, ensuring data integrity and security. Communication between the client and backend remained consistent with low latency under normal usage.

The machine learning based question paper generation module performed as expected. Based on the user input, the system correctly extracted the examination year, validated the subject domain, and generated structured question papers following the specified section-wise constraints. The ensemble learning approach using Random Forest and XGBoost effectively ranked questions using prediction scores, enabling the selection of relevant and diverse questions.

The large language model (LLM) component successfully generated responses for user queries through the /ask endpoint. Device aware model loading ensured stable execution across Compute Unified Device Architecture (CUDA), Metal Performance Shaders (MPS), and CPU environments. Loading the model during application startup significantly reduced request latency. Overall system response time remained acceptable, indicating suitability for small to medium-scale academic usage.

Only essential figures, such as API responses and generated question paper outputs, are included in this chapter. Additional logs and supporting outputs are provided in the Appendix.

4.3 Comparison with Objectives or Existing Systems

The obtained results align closely with the original project objectives. The system successfully automated the process of question paper generation, reducing manual effort while maintaining structure, relevance, and topic diversity. The integration of machine learning enabled intelligent question selection rather than static or rule-based generation.

Compared to existing traditional systems discussed in Chapter 2, which primarily rely on manual preparation or fixed question banks, the proposed system demonstrates clear improvements through predictive modeling and similarity-based deduplication. While some platforms offer basic automation, the use of ensemble learning and semantic filtering differentiates the proposed approach. Minor deviations from the initial plan include support for only a single subject (C Programming) and the absence of real-time model retraining. These limitations arose due to dataset availability and computational constraints but did not affect the core functionality of the system.

4.4 Challenges and Limitations

Several challenges were encountered during project implementation. Preparing and structuring the dataset for machine learning training was time-consuming and limited the diversity of available questions. Integrating the machine learning pipeline with the FastAPI backend required careful handling of model persistence and input validation.

Performance constraints were observed during LLM initialization, particularly on CPU-only systems. This issue was mitigated by loading the model at server startup and using device-specific configurations. Large-scale concurrent usage was not tested due to hardware and deployment limitations.

Some limitations remain, including dependency on dataset quality, limited subject coverage, and slower inference speed on non-GPU systems. These constraints were partially addressed through optimized API design and efficient model execution.

4.5 Discussion

The results demonstrate that the proposed system is an effective solution for automating examination question paper generation. The combination of machine learning and structured constraints enabled the generation of relevant, diverse, and non-duplicated questions with minimal manual intervention.

Although certain limitations exist, the system meets its primary objectives and validates the feasibility of using machine learning and language models in academic assessment automation. The findings suggest that future enhancements, such as multi-subject support, larger datasets, and improved deployment scalability, could further enhance system performance and practical applicability.

Chapter 5

Conclusion and Future Works

NepLearn: Smart Learning for Every Student was successfully designed and implemented to address the challenge of unstructured and generic study resources faced by undergraduate students. The project achieved its primary objective of developing a machine learning–based platform capable of organizing academic questions, identifying patterns from historical exam data, and providing personalized question recommendations.

The system effectively integrates question clustering using K-Means and DBSCAN with predictive modeling using Random Forest and XGBoost to support intelligent question selection. In addition, the integration of a TinyLlama-based chatbot enabled interactive query handling and conceptual support, enhancing the overall learning experience. A secure web-based platform was implemented using React.js and FastAPI, providing user authentication and seamless interaction between the frontend, backend, and machine learning components.

Overall, the project demonstrates the feasibility and usefulness of applying machine learning techniques to educational recommendation systems. The system meets its core objectives by improving content organization, supporting targeted exam preparation, and reducing the effort required by students to identify relevant practice questions. The work also provided valuable practical experience in integrating machine learning models, web technologies, and conversational AI into a unified system.

5.1 Limitations

Despite achieving its objectives, the project has several limitations that should be acknowledged:

- The system was trained and evaluated on a limited dataset consisting mainly of C programming questions, which may restrict the generalization of recommendations to other subjects.
- Model performance is dependent on the quality and diversity of the available question data, and the current dataset size limits deeper pattern discovery.
- Scalability under heavy concurrent usage was not extensively tested due to time and resource constraints.
- The chatbot interaction does not maintain conversation history, as each request is processed independently.
- Real-time performance optimization and deployment in a large-scale production environment were not explored.

These limitations arose primarily due to constraints related to time, data availability, and project scope.

5.2 Future Enhancements

Several enhancements can be considered to overcome the identified limitations and extend the scope of the system in the future:

- Expanding the dataset to include multiple subjects and academic levels to improve model generalization and recommendation accuracy.
- Incorporating more advanced recommendation techniques, such as deep learning–based ranking models or hybrid recommendation approaches.
- Enhancing chatbot capabilities by maintaining conversational context and providing more detailed explanatory responses.
- Improving system scalability through optimized deployment strategies and load handling.
- Deploying the system in real academic settings to collect user feedback and further refine recommendation quality.

These improvements feel like the next step and would make the system more effective and supportive as an intelligent educational platform.

References

Boateng, G., John, S., Glago, A., and Boateng, S. (2022). Kwame for science: An ai teaching assistant based on sentence-bert for science education in west africa.

Dharmadhikari, S. and Kamat, S. (2023). Eklavvya: Ai-powered question bank and assessment analytics platform.

Appendix A

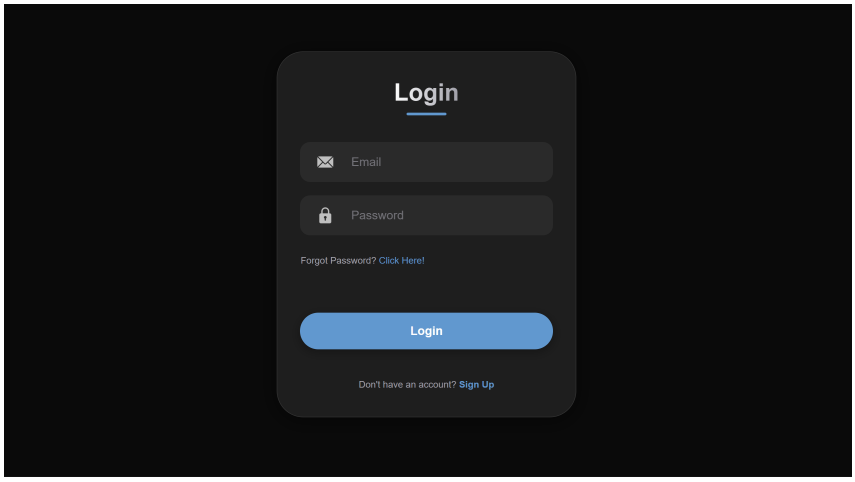

Figure 5.1: Login page

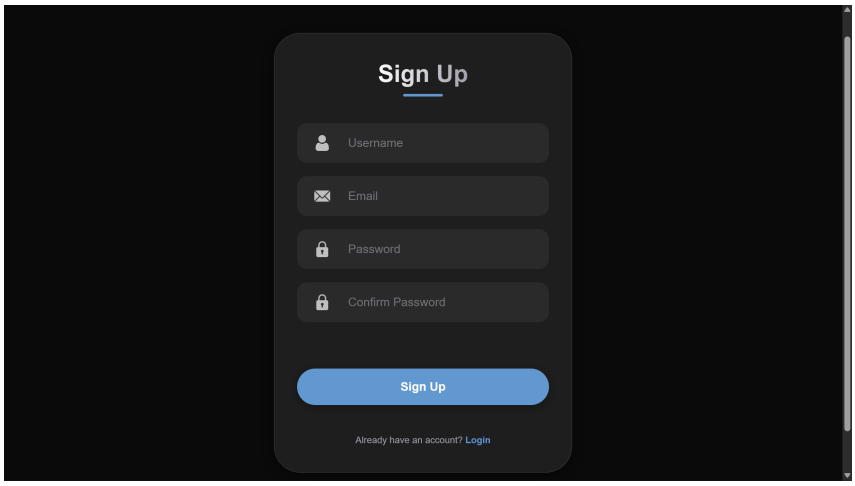

Figure 5.2: Signup example

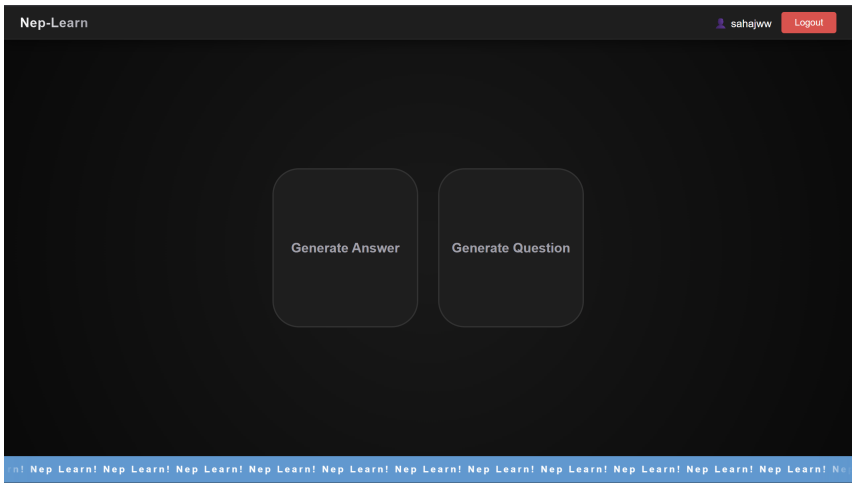

Figure 5.3: Home page

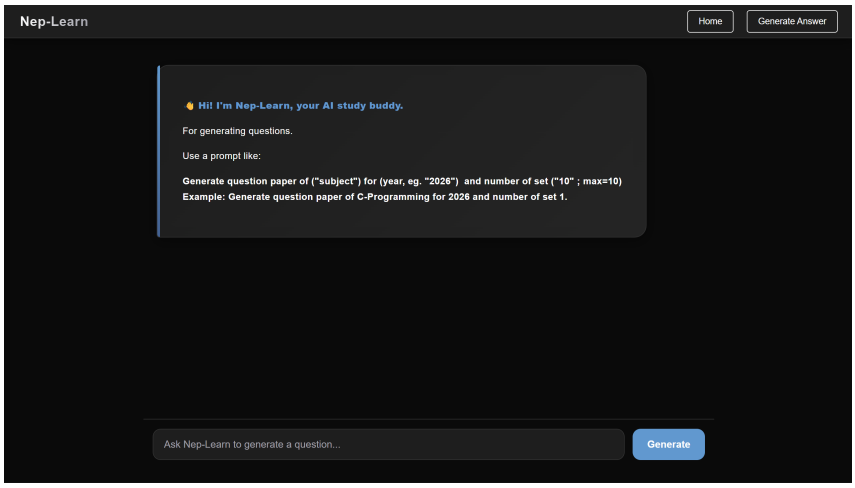

Figure 5.4: Generate question page

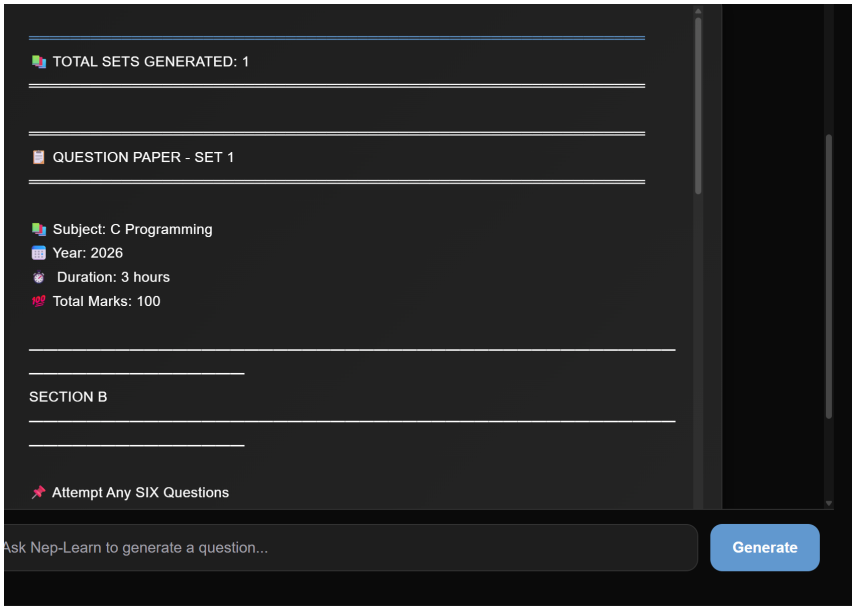

Figure 5.5: Question generation example

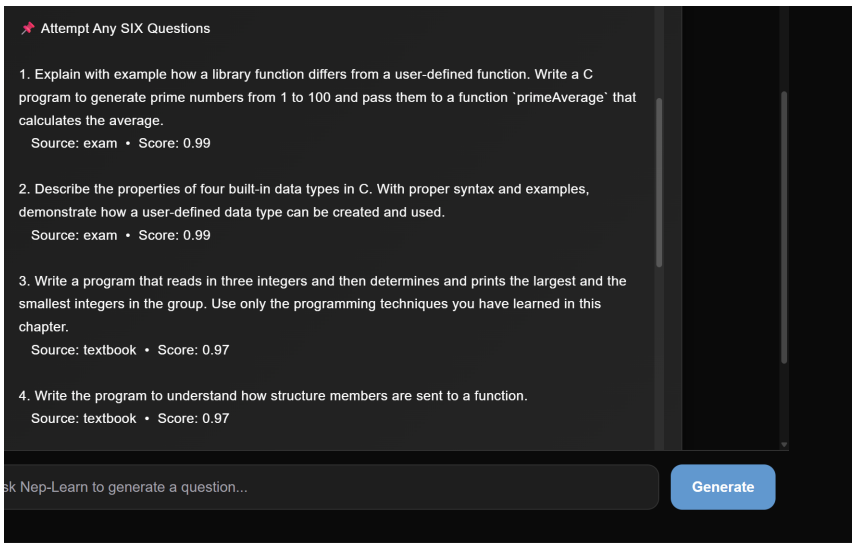

Figure 5.6: Question generation example 2

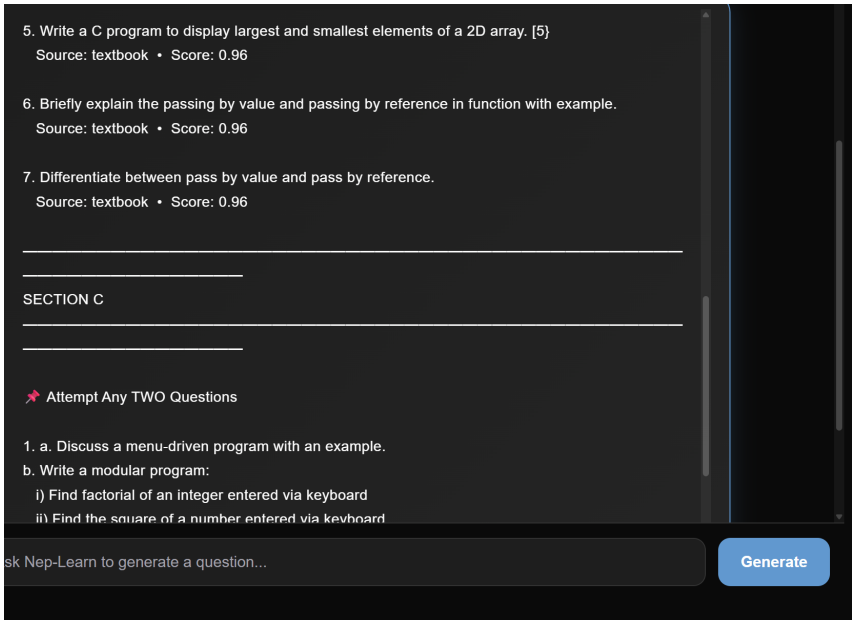

Figure 5.7: Question generation example 3

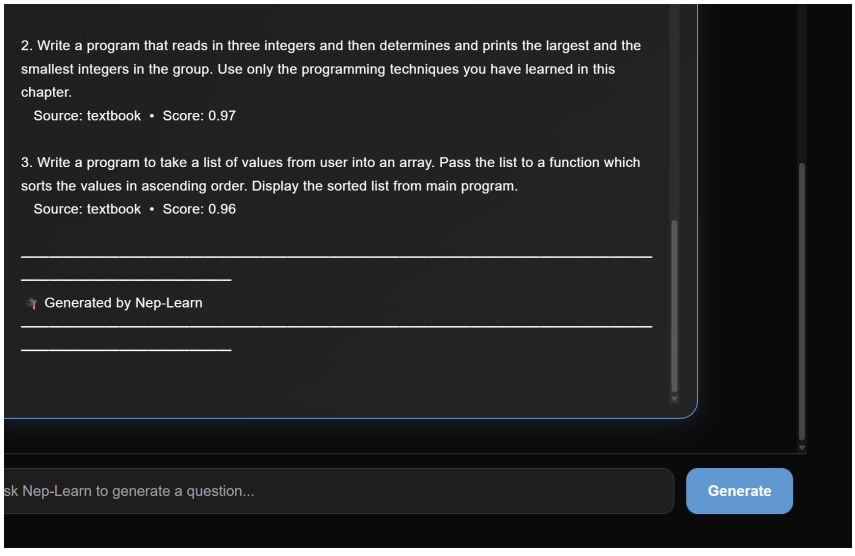

Figure 5.8: Question generation example 4

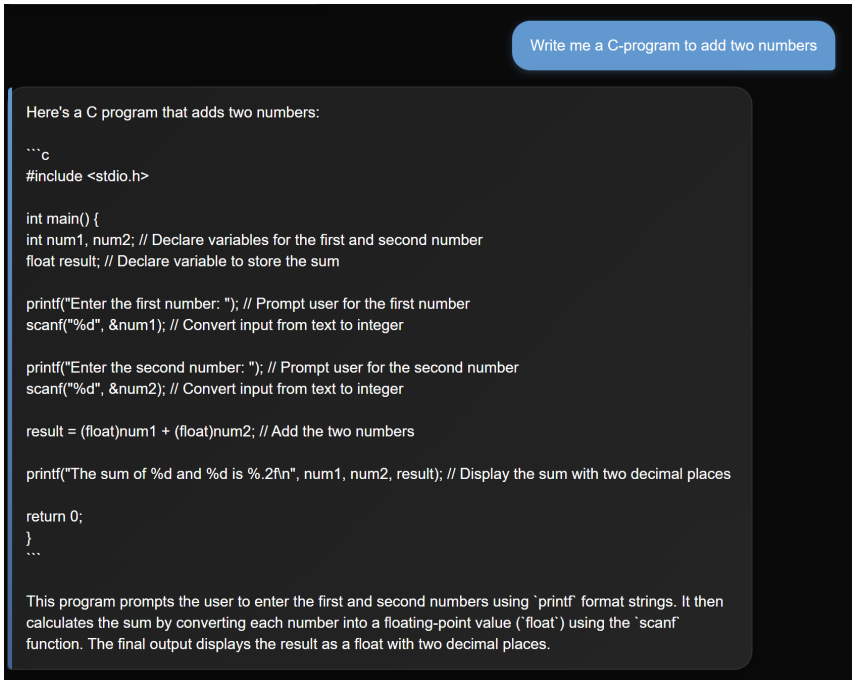

Figure 5.9: Answer generation example