TRIBHUVAN UNIVERSITY

INSTITUE OF SCIENCE AND TECHNOLOGY

BHAKTAPUR MULTIPLE CAMPUS

A Project Report

On

"NLTTA: Nepali Language Transliteration and Translation Automation"

Under the Supervision of
Mr. Sushant Paudel
Department of CSIT
Bhaktapur Multiple Campus

Submitted To:
Office of the Dean
Institute of Science and Technology, Tribhuvan University
Kirtipur, Nepal

In partial fulfillment of the requirement of Bachelor of Science in Computer Science & Information Technology

Submitted By:

Bibek Rawal (28294/078)

Prabesh Gautam (28304/078)

Rahul Koju (28312/078)

November, 2025

BHAKTAPUR MULTIPLE CAMPUS

A Constituent Campus of Tribhuvan University

SUPERVISOR'S RECOMMENDATION

I hereby recommend that this project prepared under my supervision by **Bibek Rawal, Prabesh Gautam and Rahul Koju** on the topic "**NLTTA: Nepali Language Translation and Transliteration"** in partial fulfillment of the requirement for the degree of BSc. in Computer Science and Information Technology (BSc. CSIT) be processed for evaluation.

Mr. Sushant Paudel

Supervisor

Department of Computer Science and IT

Bhaktapur Multiple Campus

BHAKTAPUR MULTIPLE CAMPUS

A Constituent Campus of Tribhuvan University

CERTIFICATE OF APPROVAL

This is to certify that this project prepared by **Bibek Rawal [28294/078], Prabesh Gautam [28304/078], Rahul Koju [28312/078]** entitled "**NLTTA: Nepali Language Translation and Transliteration Automation**" in partial fulfillment of the requirements for the degree of BSc. CSIT has been well studied. In our opinion, it is satisfactory in the scope and quality of the required degree.

Mr. Sushant Paudel
Supervisor
Department of CSIT
Bhaktapur Multiple Campus, TU

Mr. Sushant Paudel
Program Co-Ordinator
Department of CSIT
Bhaktapur Multiple Campus, TU

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

We would like to express our deepest appreciation to all those who provided us the possibility to complete this report. A special gratitude we give to our project supervisor, **Mr. Sushant Paudel**, in whose contribution stimulating suggestions and encouragement, helped us to coordinate our project especially in writing this report. We extend our sincere appreciation to the staff of Bhaktapur Multiple Campus for their crucial support, granting us permission to use all the necessary equipment and materials required to complete this project. We were also fortunate to receive the constant support and guidance from our seniors and the entire teaching staff of the B.Sc. CSIT Department, which was instrumental in the successful completion of our work. Our thanks also go to the non-teaching staff of the CSIT department for their timely and invaluable assistance. We particularly value the constructive criticism and advice given by our supervisor and the project presentation panel members; their comments significantly improved our final presentation skills. This project would not have been possible without the kind help of every one of these individuals, and we sincerely appreciate their contributions.

With respect,

Bibek Rawal (28294/078)

Prabesh Gautam (28304/078)

Rahul Koju (28312/078)

ABSTRACT

This project presents a complete pipeline for translating Romanized Nepali queries into English and retrieving the most relevant news articles using a fine-tuned mBART model called NLTTA and BM25 retrieval algorithm. We fine-tuned the multilingual mBART model on a parallel corpus of Romanized Nepali and English sentences to enhance its translation accuracy for Romanized (transliterated) languages, low-resource language input. The fine-tuned model is deployed via a Django-based REST API for backend and NextJS for frontend, which serves translation functionality in real time. Once a Romanized Nepali query is translated into English, the system applies the BM25 ranking algorithm on a pre-indexed English news database to retrieve and rank the most relevant articles based on the query. This hybrid system bridges the linguistic gap for Nepali speakers, allowing effective English news search and information access through a robust combination of neural translation and probabilistic information retrieval.

Keywords: BM25, mBART, Neural Translation, pre-indexed, Probabilistic retrieval, REST API

TABLE OF CONTENTS

LIST OF ABBREVATIONS

API	Application Programming Interface
BART	Bidirectional and Auto-Regressive Transformer
BERT	Bidirectional Encoder Representations from Transformers
BLEU	Bilingual Evaluation Understudy
BM25	Best Matching 25
CNN	Convolutional Neural Network
CPU	Central Processing Units
DL	Deep Learning
E-commerce	Electronic Commerce
GPU	Graphical Processing Units
IDF	Inverse Document Frequency
JSX	JavaScript XML
LLM	Large Language Model
mBART	Multilingual Bidirectional and Auto-Regressive Transformer

ML	Machine Learning
MuRIL	Multilingual Representations for Indian Languages
NLTTA	Nepali Language Translation and Transliteration Automation
MYSQL	My Structured Query Language
NMT	Neural Machine Translation
REST	Representational State Transfer
RNN	Recurrent Neural Network
Seq2Seq	Sequence to Sequence
SSG	Static Site Generation
SSR	Server-Side Rendering
TC	Test Case
TF	Term Frequency
TF-IDF	Term Frequency–Inverse Document Frequency
TPU	Tensor Processing Units
UI	User Interface

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

1.1 Introduction

In Nepal, many people commonly communicate using a mixture of Nepali (both in Devanagari and Romanized scripts) and English, especially in informal digital contexts such as chatting, searching, and social media. Nepali is spoken by over 33 million people worldwide including both native and second-language speakers. However, most existing digital platforms and search engines are not designed to understand such code-mixed or informal queries. This creates a significant barrier for many users, especially those who are partially literate or not fluent in English. Thus, making accessing relevant and meaningful information online a hassle.

Neural machine translation (NMT) [1] is known to give state-of-the-art translations for a variety of language pairs. NMT is known to perform poorly for language pairs for which parallel corpora are scarce. This happens due to lack of translation knowledge as well as due to overfitting which is inevitable in a low-resource setting. Fortunately, transfer learning via cross-lingual transfer [2] can significantly improve translation quality in a low-resource situation.

To address this challenge, we introduce NLTTA (Nepali Language Translation and Transliteration Automation), a deep learning-based text understanding model specifically designed to process Nepali language in both Devanagari and Romanized forms, as well as mixed English-Nepali inputs. NLITA can transliterate Devanagari to Romanized Nepali translate text between Nepali and English, and understand informal, chat-style user inputs. The system aims to improve digital accessibility and enhance the usability of applications such as news search engines or product search on E-commerce sites for Nepali users. Through the integrating of this translation and transliteration system, the news platform delivers more accurate and inclusive search results, enhancing user experience and accessibility for a multilingual audience.

1.2 Problem Statement

In Nepal, a significant portion of the population is only partially literate or has limited fluency in English. As a result, many users rely on informal communication using Romanized Nepali

(e.g., “sansad” > “parliament”, “छात्रवृत्ति” > “scholarship”) or mix English with Nepali while typing. Unfortunately, current search engines and digital systems are not equipped to understand these types of informal or bilingual inputs.

This leads to several problems:

- Users receive irrelevant or no results when using Romanized or mixed-language queries.
- Nepali-speaking users face difficulty in accessing online information effectively.
- People with low English proficiency are digitally excluded.

Therefore, there is a strong need for a system that can:

- Understand both Devanagari and Romanized Nepali.
- Accurately transliterate and translate between Nepali and English.
- Handle informal inputs to return relevant results.

1.3 Objectives

This project aims to develop a model that understands Nepali in both Devanagari and Romanized scripts, handles informal mixed-language inputs, and improves access to digital content for users with limited English proficiency and integrate it with a web application for searching news contents.

The main objectives of this project are:

- To fine tune mBART model that can understand and process text in Nepali (both Devanagari and Romanized forms).
- To integrate this system into a news search application that allows users to search using natural, informal Nepali.
- To implement BM25 algorithm to retrieve the most relevant news.

1.4 Scope and Limitation

NLTTA model addresses a critical linguistic barrier in Nepal by enabling users to interact with digital systems using Devanagari Nepali, Romanized Nepali, and mixed-language (English-Nepali) inputs. It allows users especially those with limited English proficiency or partial literacy to search and access content more effectively, making it particularly impactful in rural communities. The system is designed to process informal, chat-style inputs and return accurate translations and relevant news results. NLTTA can be integrated into various domains like news

search platforms, e-commerce, and public service portals. In the developing digitization of Nepal, NLTTA ensures inclusive access to information and enhances digital participation for a broader population segment.

The limitations of the project may include:

- The system's translation accuracy relies on the quality and diversity of the training corpus, which may need further enhancement to fully support complex informal expressions.
- Real-time performance may be limited in low-bandwidth areas, especially during model inference or API communication.
- Romanized Nepali lacks standardization, which can introduce variability and challenges in accurately interpreting user intent.
- The fine-tuned models, while effective, require ongoing updates and domain-specific tuning to maintain relevance and accuracy over time.

1.5 Development Methodologies

The development of the NLTTA followed the Agile methodology, a flexible and iterative approach that supports frequent feedback and continuous improvement. Unlike traditional models, Agile allows adapting to evolving requirements, which was crucial for this project since accuracy, usability, and user feedback were essential for refining the system. This approach was essential because it allowed the team to adapt to evolving requirements, focusing on accuracy, useability and user feedback to refine the system effectively.

The process began with a two-week planning phase, where the project scope and main goals, including the translation, transliteration, and a simple user interface design, were clearly defined along with research on the model architecture. The subsequent twelve-week development phase was divided into four sprints: the first sprint focused on scarping data to build an appropriate dataset for training and testing whereas the second involved selecting and fine tuning the chosen model. The third sprint was dedicated to designing the MySQL database and building the BM25 retrieval system followed by the fourth sprint where all the components were integrated into the web application. Finally, a two-week testing and documentation phase

was conducted, which included BLEU and BERT score evaluations and preparations of comprehensive documentation to ensure easy use of the system.

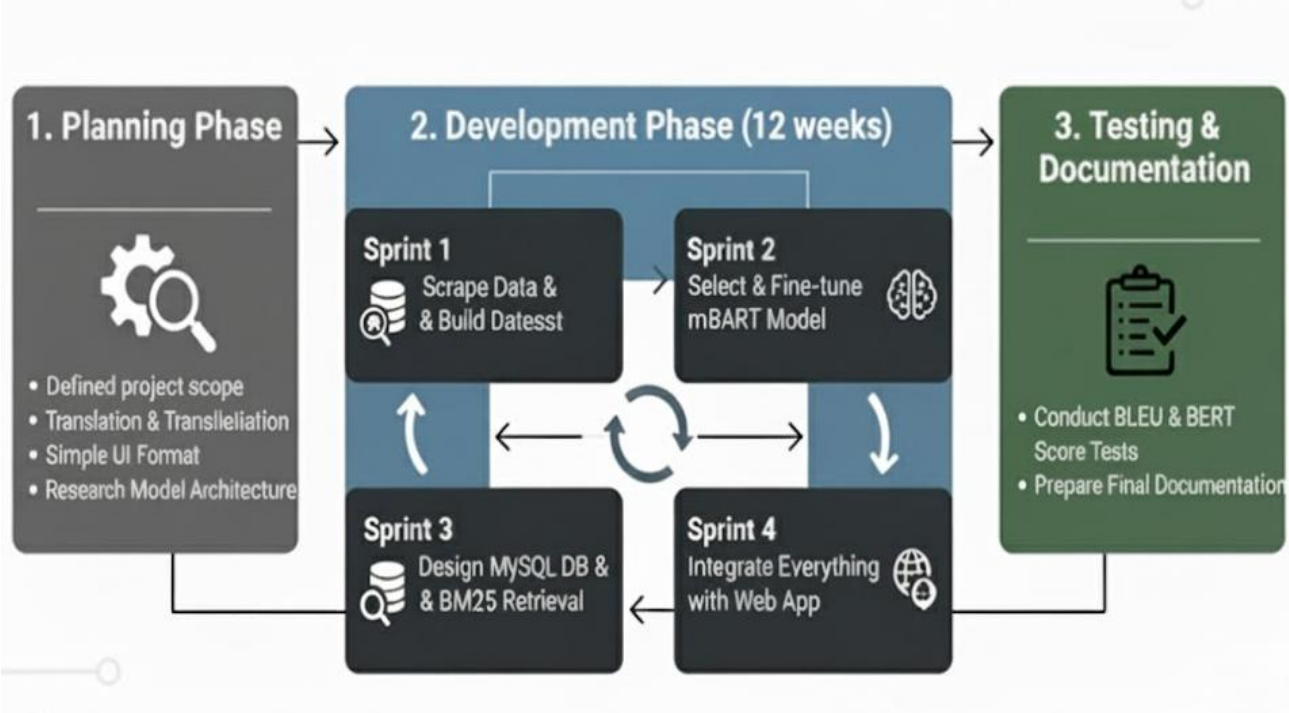

Figure 1.1: Agile Development Process of NLTTA

Due to the complexity of the application the development process took a lot of refinement of the previous process and collaborative building of different components. Thus, agile methodology was selected as it allowed the best implementation of parallel development through different sprints.

1.6 Report Organization

This report is structured into six comprehensive chapters that collectively document the design, development, and evaluation of the NLLTA system. This report is structured into six comprehensive chapters that collectively document the design, development, and evaluation of the Nepali Language Translation and Transliteration Automation (NLLTA) system. The **Chapter 1,** Introduction, contains the problem statement, objectives, scope, limitations, methodologies and the significance of developing a robust translation and transliteration system for the Nepali language. Similarly, the **Chapter 2,** consists of the background and the literature review of this report. This chapter details the development of others in the field of

this exact problem or similar opportunities and the possibilities for NLTTA based on lesson learned from others. The **Chapter 3,** consists of functional and non-functional requirements, feasibility study and use case analysis. This chapter shows the main features of NLTTA provides and how the users interact with the system. In addition, **Chapter 4,** focuses on the system's architecture, module decomposition, database schema, workflow diagrams, user interface design considerations and algorithm design for translation and retrieval. The chapter takes a deep dive into how different components are built and how they all work other to make NLTTA an efficient system. Likewise, **Chapter 5,** outlines the development environment, tools, and frameworks used along with different testing approaches. It goes in depth of the tests done and the metrics used to evaluate the performance of NLTTA with respect to other systems. The final **Chapter 6,** summarizes the key outcomes, reflects on the effectiveness of the developed system, and highlights possible improvements and research directions. It focuses on what the current system lacks and how the future versions of NLTTA or similar projects can perform even better.

The final part of the report consists of **References** and **Appendices**. The references are listed in accordance to the IEEE referencing standards and the Appendices includes the screenshots of the system and the major source code snippets.

CHAPTER 2: BACKGROUND STUDY AND LITERATURE REVIEW

2.1 Background Study

The development of the web application for retrieving by translation of Romanized and mixed Nepali involves several fundamental theories, concepts, and terminologies. These include:

2.1.1 Deep Learning:

DL is a subset of machine learning that uses neural networks with multiple layers to model and understand complex patterns in data. From the many deep learning models, we are using transformer architectures for translating Nepali and handling mixed-language queries. These models learn contextual relationships across words and scripts, enabling more accurate translations and understanding of informal inputs.

2.1.2 Transformers:

Transformers are a deep learning architecture based on self-attention mechanisms. Unlike earlier sequential models like RNNs, transformers process entire sequences in parallel, capturing long-range dependencies and contextual relationships efficiently. It excels in sequence-to-sequence tasks like translation and text generation by capturing relationships between elements in a sequence.

2.1.3 mBART:

mBART is a sequence-to-sequence language model developed by Facebook AI for multilingual translation tasks. It is trained on multiple languages simultaneously and can handle translation and text generation across diverse language pairs. mBART is foundationally built upon BART model but using tags for support of different languages. It uses 12 layers of encoder and decoder of the Transformer architecture.

2.1.4 Web Applications:

Web-based platforms that enable users to perform specific tasks through an internet browser. They are highly accessible and scalable, making them ideal for deploying AI-powered tools to users in remote areas. We are designing and launching the web application using Django framework developed in Python programming language for backend and NextJS for frontend.

2.1.5 BM25:

BM25 is a ranking algorithm used in information retrieval to determine the relevance of documents to a given search query. It improves upon earlier models like TF-IDF by considering term frequency saturation and document length normalization, making it highly effective for ranking search results. BM25 recognizes that the impact of term frequency plateaus after a certain point. Adding more occurrences of a term doesn't proportionally increase the document's relevance score.

2.2 Literature Review

Processing bilingual or code-mixed languages such as Nepali and English presents significant challenges, especially due to the low-resource nature of the Nepali language. Various works have attempted to address similar challenges in related languages, particularly in South Asian contexts, using transformer-based models and multilingual pre-training approaches.

One of the most relevant works is MuRIL [2], was developed using a transformer-based language model tailored for Indian languages including Hindi, Tamil, and others. It used transliterated Roman script data along with native script data to improve understanding across diverse linguistic forms. This is directly relevant to the work on Nepali, where Romanized Nepali is common in informal contexts. MuRIL demonstrated that pre-training using both native and Romanized scripts significantly enhances downstream task performance, especially for sentiment analysis and question answering.

mBART [3] introduced a sequence-to-sequence pre-trained model for multilingual translation tasks. This model enabled better cross-lingual understanding and translation by using a denoising objective during pre-training. Its ability to generalize across language pairs without requiring large parallel datasets made it suitable for low-resource language applications like Nepali-English.

The SentencePiece tokenizer [4], is a language-agnostic subword tokenizer that enables better vocabulary representation in neural machine translation systems. This is especially important for morphologically rich and agglutinative languages such as Nepali, where word-level tokenization can be inefficient.

Transformers [5] serve as the base for all modern NLP models including BERT, mBART, and MuRIL. Its attention mechanism allows models to capture long-range dependencies, essential for effective translation and semantic understanding.

In terms of information retrieval, traditional methods like TF-IDF and BM25 have proven effective in indexing and ranking documents. While neural methods like dense embeddings [6] are gaining traction, in this project, we choose to implement TF-IDF due to its simplicity, interpretability, and effectiveness in keyword-based document search. It enables fast and relevant retrieval of news articles based on bilingual (translated and transliterated) user queries.

NepaliBERT [7], is a recent advancement that trained a masked language model specifically on a Nepali corpus. While it focuses only on Devanagari script Nepali, it demonstrates the effectiveness of fine-tuning transformer models on Nepali text. However, it does not handle Romanized Nepali or bilingual input, which limits its application in real-world scenarios like search or chat-based interfaces.

2.2.1 Key Insights for the Project

- The reviewed literature confirms that transformer-based models like mBART are highly effective for multilingual tasks, particularly for low-resource languages when fine-tuned on domain-specific datasets.
- Transformers capture context well: Their attention mechanism helps translate mixed or informal bilingual inputs accurately.
- BM25 remains practical: Despite neural advances, BM25 offers fast, interpretable document retrieval suitable for real-time search.

This review supports the feasibility of the proposed project and provides valuable insights for overcoming potential challenges, ensuring the solution is practical and impactful.

CHAPTER 3: SYSTEM ANALYSIS

3.1 Requirement Analysis

3.1.1 Functional Requirements

Functional requirements describe the core functionalities of the web application. Below are the key functional requirements illustrated using a use case description.

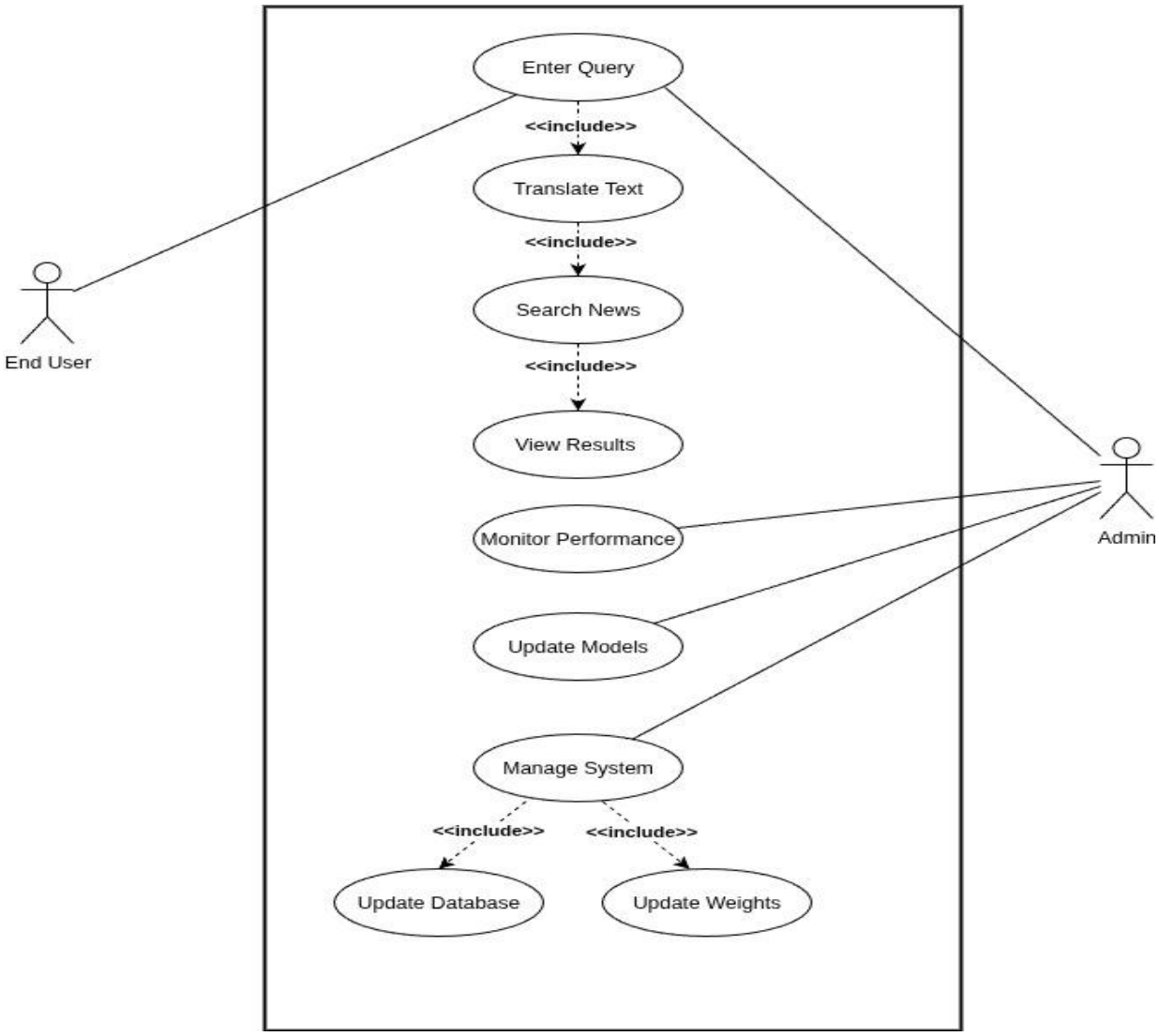

Figure 3.1 Use Case Diagram for Searching and Retrieval of Relevant News

The system should be able to provide users with the most relevant news according to their queries in both Romanized Nepali and English. The process starts when users enter a query in Romanized Nepali. The full retrieval process is dynamically set into motion the moment a user

enters a query in Romanized Nepali. The query then triggers the translation process to get the proper English query.

This input immediately moves through a sophisticated, multi-stage pipeline: first, it passes through the translation engine (like the NLTTA model) where it is converted into the target language used by the document collection (e.g., English). Subsequently, the translated query enters the ranking algorithm phase, typically utilizing the BM25 model, which calculates relevance scores by comparing the query against all indexed news articles. Finally, the indexes of the top-ranked documents are used in the database retrieval phase to quickly fetch the complete articles from the database.

3.1.2. Non-Functional Requirements

Non-functional requirements describe the system's overall qualities and constraints.

a. **Performance**: The system should process any query and provide relevant news within 3 seconds. The relevant news retrieved must have accuracy rate above 80%.

b. **Scalability**: The system should handle concurrent users and support the addition of news updates along with model fine tuning for future.

c. **Accessibility**: The web application should be optimized for mobile devices and available in multiple languages, including Nepali.

d. **Usability**: The interface should be user-friendly, requiring minimal technical knowledge for operation.

e. **Maintainability**: The system should allow easy updates to the transformer model and software components on both frontend and backend.

f. **Connectivity**: The application should function effectively in areas with limited bandwidth by optimizing image uploads and system response times.

These requirements form the foundation for designing and implementing a robust, user-centric disease detection system tailored for Nepalese farmers.

3.2 Feasibility Analysis

3.2.1. Technical Feasibility

The project utilizes a transformer-based deep learning models such as mBART as the base, which are known for handling cross-lingual tasks. These models will be fine-tuned on a Nepali-English parallel corpus (extracted from Wikipedia dumps and semi-manually annotated Romanized Nepali data). The biggest technical hurdle would be the resource to train such a model which is solved by utilization of checkpoints saving. The checkpoint feature will allow us to utilize Google Colab provided resources to train and test the model for parts of data everyday over the period of the project.

3.2.2. Operational Feasibility

The translation model is deployed via an API in Python which is integrate into the news searching platform. The API is developed in Python using Django framework, with the model running on GPU-supported environments or CPU if the GPU memory is limited. The news platform is made using Django for backend and NextJs for the frontend.

3.2.3. Economic Feasibility

The project aims to utilize freely and publicly available platform like Google Colab for training compute and publicly available datasets, annotated semi-manually to minimize cost. The project utilizes the pre-trained models thus making the cost of the project minimal. Any future deployment could be scaled using cloud platforms like Azure.

3.2.4. Schedule Feasibility

The time allocated for the completion of this project was a whole semester. So, the project had enough time for completion. The project includes training a LLM so it will take more time than usual. The project is aimed to be completed within 4 months leaving a month of time as grace period for unforeseeable circumstances. Hence, the project has been developed according to the following time schedule to make our application schedule feasible:

Table 3.1: Schedule Table

S.No.	Task Name	Duration	Preceding Task
1	Literature Review and Data Collection	1 week	0
2	Preprocessing	1 week	1
3	Model Selection	1 week	2
4	Training and Fine Tuning	9 weeks	3
5	Web App Development	7 weeks	3
6	Document Indexing	3 weeks	5
7	API Development and Integration	2 weeks	4,6
8	Testing and Re-adjustment	1 week	7
9	Final Report and Documentation	1 week	8

Gantt Chart:

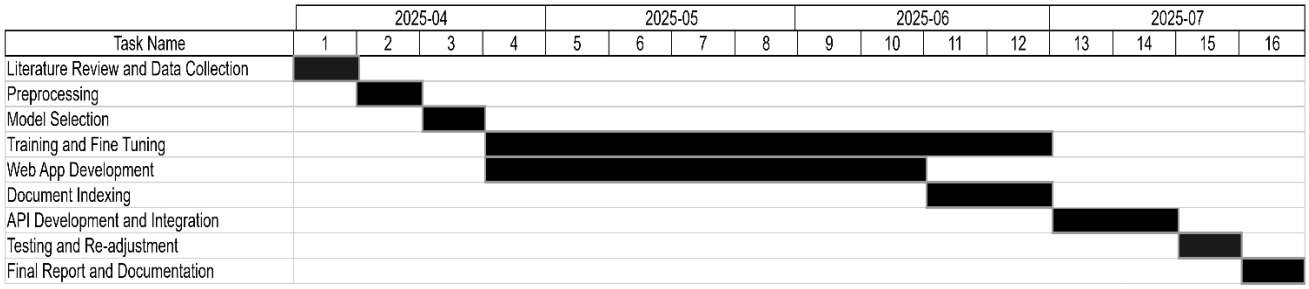

Figure 3.2: Gantt Chart of Time Schedule

3.3 Dataset Description

The dataset consists of a combination of two databases. The first is a Devanagari Nepali and English parallel corpus scrapped from the Wikipedia chunks and refining the publicly distributed chunks by Wikipedia. The second dataset is a combination of open-source datasets of parallel corpus of Devanagari Nepali and English. The Devanagari Nepali from the final combined dataset of 40,000 samples is converted to Romanized Nepali using the Transliteration module of NLTTA using rule-based Indic transliteration.

Our final dataset is the parallel corpus of Devanagari Nepali, Romanized Nepali and English sentence consisting of about 40,000. The dataset was spilt 70% for training and 20% and 10% for validation and test.

```
Dataset loaded successfully!
Total rows: 40000
Columns: ['Unnamed: 0', 'english_sent', 'nepali_sent', 'roman_sent']
```

	Unnamed: 0	english_sent	nepali_sent	roman_sent
0	0	It happened after the death of Saul, when Davi...	दाऊदले अमालेकीहरूलाई हराएर पछि सिकलग गए। यो शा...	dAUdale amAlekIharUlAI harAera paChi sikalaga ...
1	1	it happened on the third day, that behold, a m...	तब तेस्रो दिनमा एउटा जवान सैनिक सिकलगमा आयो। त...	taba tesro dinamA euTA javAna sainika sikalaga...
2	2	David said to him, "Where do you come from?" H...	दाऊदले त्यसलाई सोधे, "तिमी कहाँबाट आयौ?" त्यस ...	dAUdale tyasalAI sodhe, "timI kahA.NbATa Ayau?...
3	3	David said to him, "How did it go? Please tell...	दाऊदले भने, "मलाई भन, के भयो?" त्यसले भन्यो, "...	dAUdale bhane, "malAI bhana, ke bhayo?" tyasal...
4	4	David said to the young man who told him, "How...	दाऊदले त्यस सैनिकलाई भने, "तिमीले कसरी जान्यौ ...	dAUdale tyasa sainikalAI bhane, "timIle kasarI...

```
Split Results:
Train set: 28000 rows (70.00%)
Validation set: 8000 rows (20.00%)
Test set: 4000 rows (10.00%)
```

Figure 3.3: Dataset Decomposition

3.4 Analysis

The application is developed using object-oriented approach.

3.4.1 Object Modeling Using Class Diagram

Class diagram in the UML is a type of static structure diagram that describes the structure of a system, it shows system's classes along with their attributes, operations and relationships among objects. The system's architecture is defined by several interconnected Classes that manage the translation, retrieval, and display of information.

The primary actors are the User and the Query class. The User class is responsible for handling user interactions and triggers the translation processes upon entering a search query. This action creates a Query object, which represents the user's original search text, its subsequent translation, and encapsulates the actual translation operations. The core language processing is managed by the Translator class, which is responsible for text normalization, tokenization, and translating the incoming text into English. Crucially, each Query is processed by one Translator for these operations.

For information storage and retrieval, the system utilizes the Feed and FeedItem classes. The Feed class represents content collections in different languages and handles their metadata and management operations. A single Feed contains multiple FeedItem objects, where each FeedItem represents an individual piece of content, such as an article or post, with specific details like the title and content.

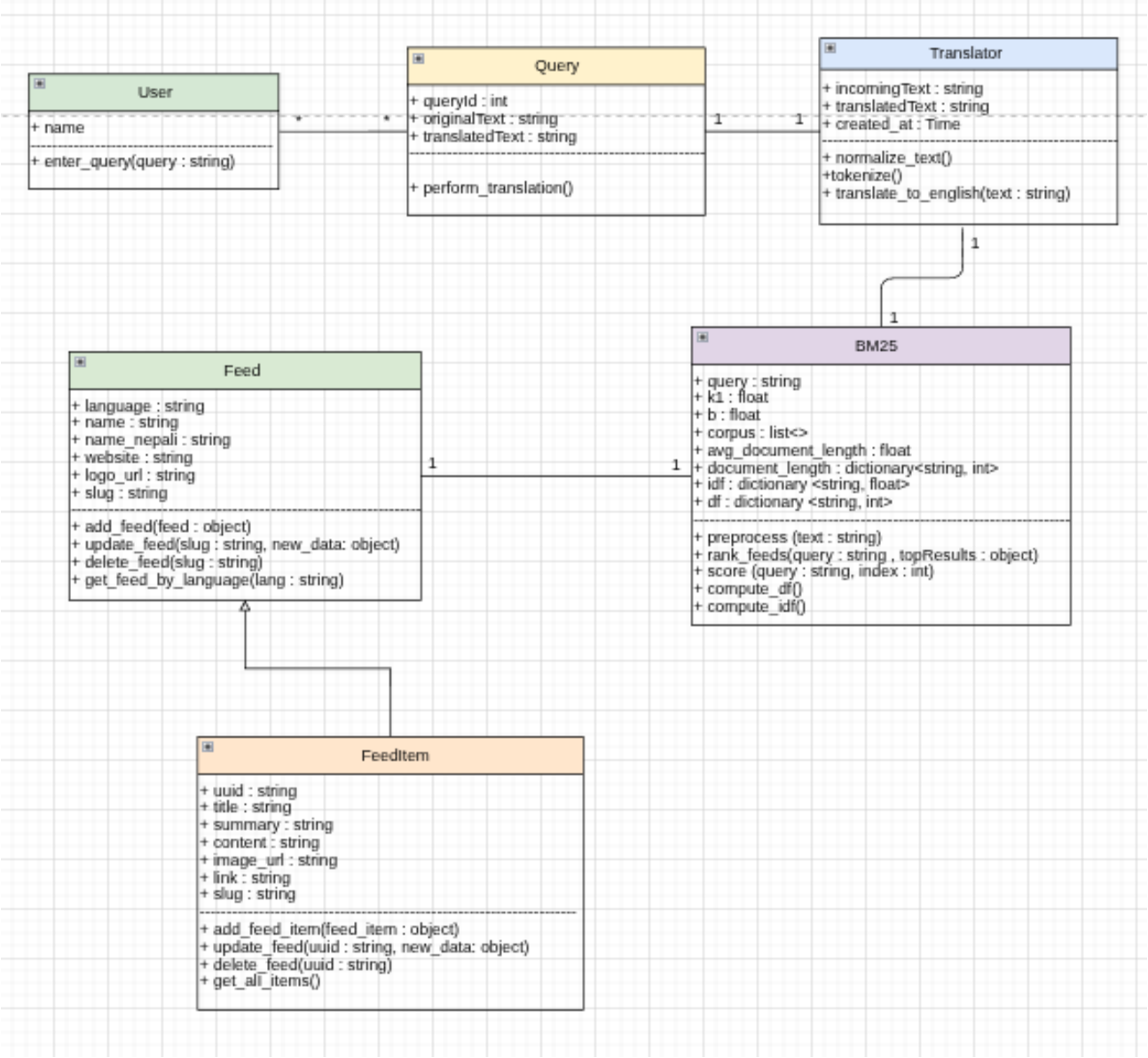

Figure 3.4: Class Diagram of NLTTA System

The retrieval intelligence is handled by the BM25 class, which implements the BM25 ranking algorithm. It contains the logic for text preprocessing and the core scoring functions necessary for information retrieval. The BM25 interacts with the Translator to preprocess and tokenize texts before ranking. Ultimately, the BM25 ranks documents in the Feed based on the translated user Query, ensuring the most relevant FeedItems are selected for display.

3.4.2 State Diagram

The system's operation is defined by a series of state transitions, beginning in the Initial State (Start) where key components are initialized: the NextJS frontend is ready, the mBART model is loaded for translation, the BM25 algorithm is prepared for ranking, and the MySQL database is connected. The process advances to the Search Input Entered state when a user provides a Romanized Nepali query, which the system validates before proceeding. This input moves to the Preprocessed Query state, where it is normalized (lowercase conversion, character removal) and formatted with the necessary language tags (e.g., >>ne_XX<<).

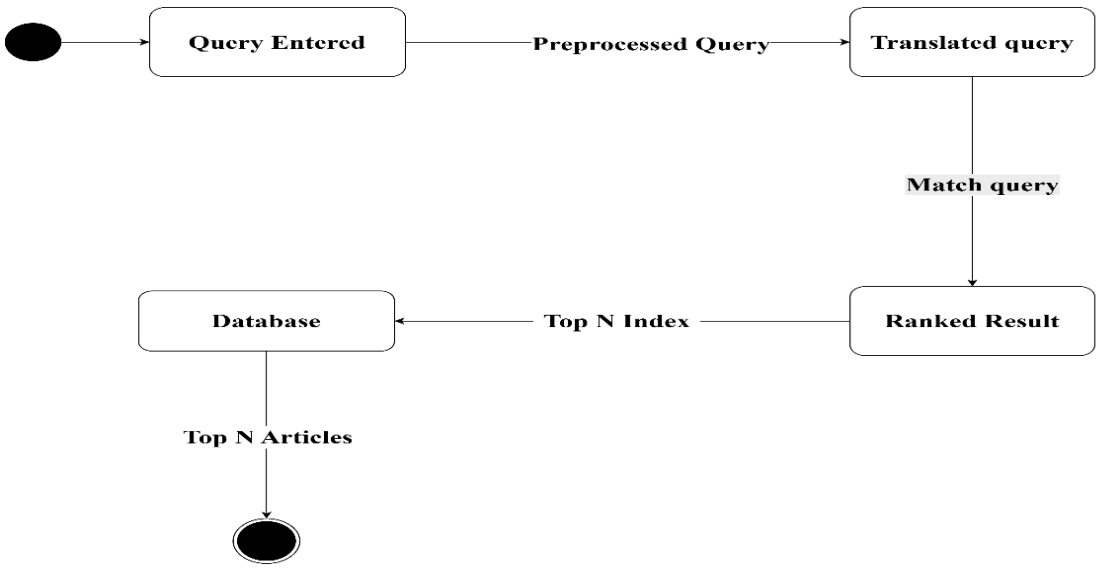

Figure 3.5: State Diagram of The System Process

The journey continues as the preprocessed input reaches the Translated Query state, where the NLTTA model uses the fine-tuned mBART model to translate the query into English (e.g., "vidharthi" → "student"). The system then enters the Rank Query state, where the BM25 algorithm calculates relevance scores for all articles and outputs a ranked list of document indices with a cutoff. These indices are used in the Database state for the MySQL database to retrieve the top N articles. These retrieved documents constitute the Top N Articles state, complete with full information and ordered by relevance. The final state is the Ranked Result, where the finished search results are organized and displayed to the user on the frontend, offering previews and interaction options.

3.4.3 Dynamic Modeling Using Sequence Diagram

Sequence diagram in the UML is a type diagram that illustrates the sequence of messages between objects in an interaction. It consists of a group of objects represented by lifelines and message they exchange during the interaction denoted by arrow symbols.

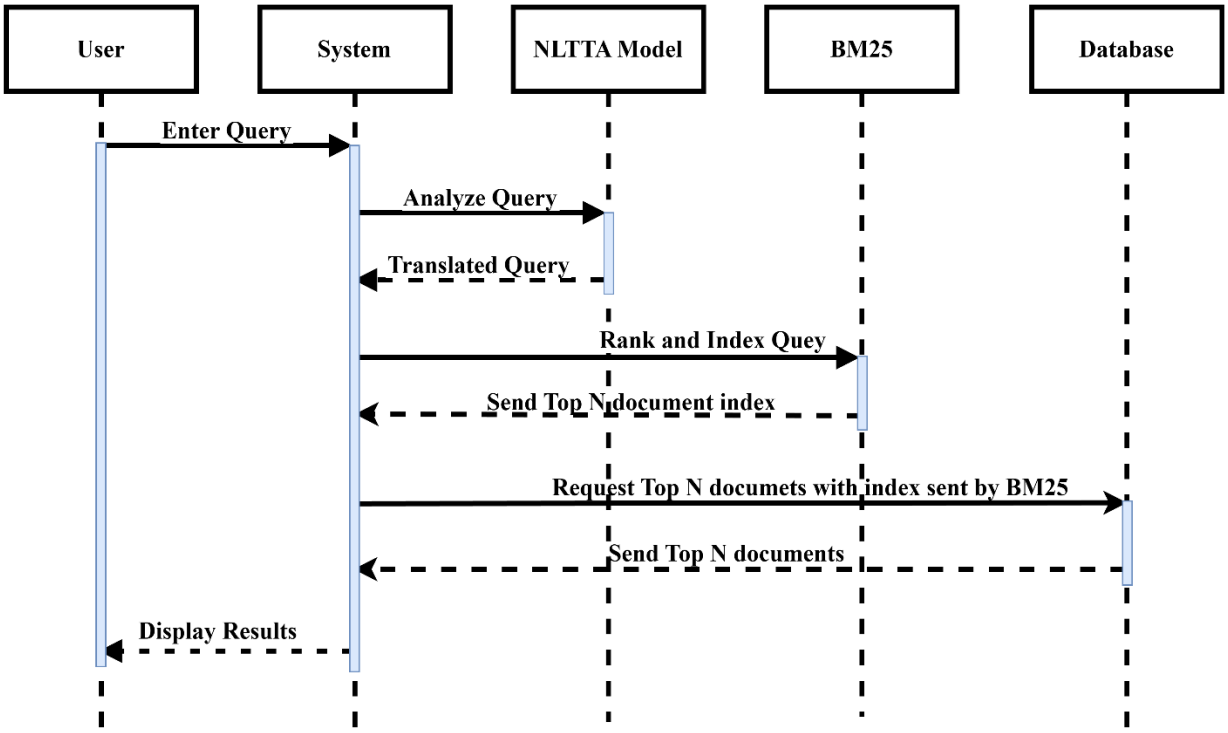

Figure 3.6: Sequence Diagram of Search and Retrieval of relevant news

In the above figure 3.4, the sequence diagram outlines the search result retrieval process, commencing when the User enters the query. The System takes this query and sends it to the NLTTA model, which performs translation and returns the processed query to the System. The System then forwards this translated query to the BM25 component. This component ranks documents based on relevance and returns the indexes of the top N documents to the System. Subsequently, the System uses these indexes to request the top N documents from the Database. After the Database sends the documents back, the System compiles and displays the final results to the User.

3.4.4 Process Modeling Using Activity Diagram

Activity diagram in the UML is a type of diagram that visually presents a series of actions or flow of control in a system. It shows workflows of stepwise activities with support for choice, iteration and concurrency.

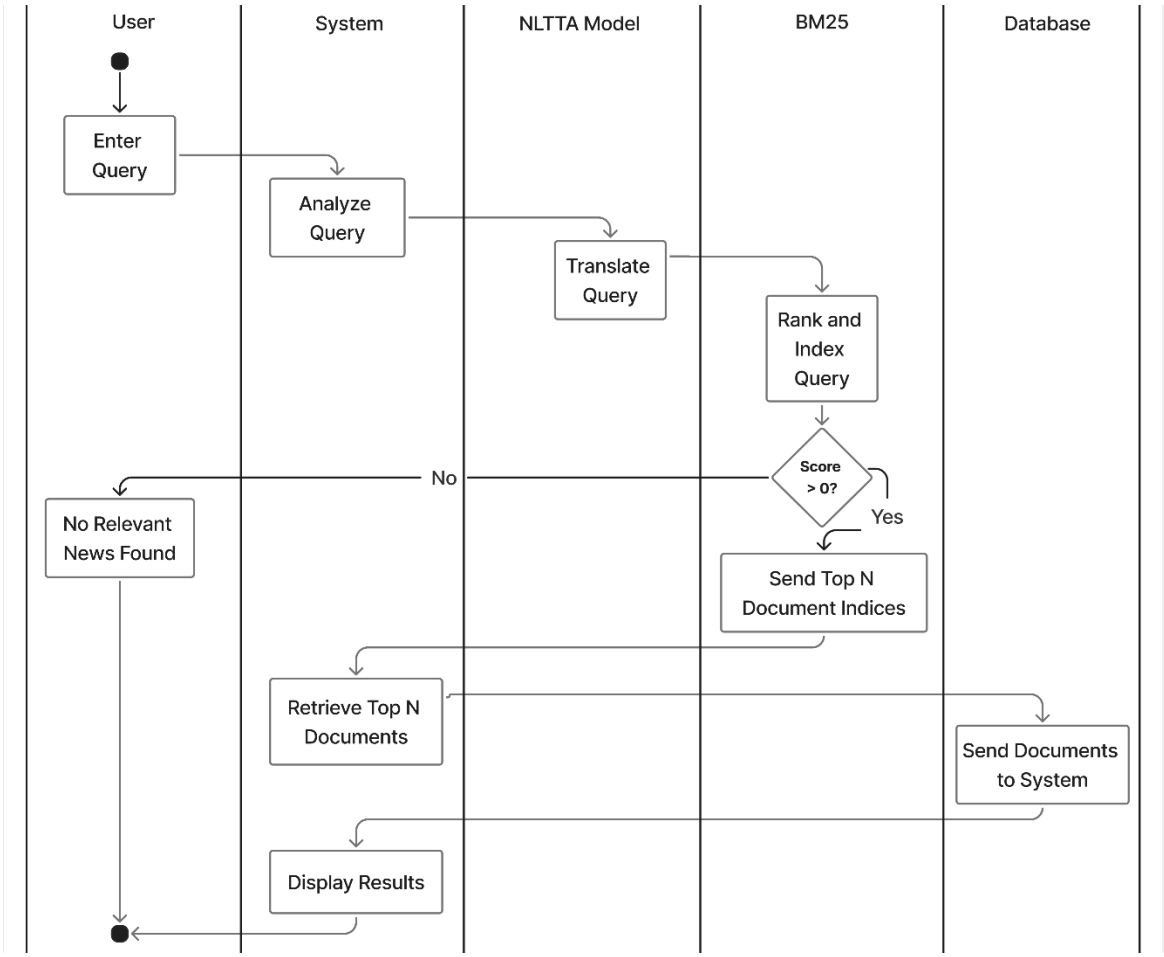

Figure 3.7: Activity Diagram of Search and Retrieval

The above Figure 3.5 below shows the activity diagram that displays the overall workflow of the query processing system involving the User, System, NLTTA Model, BM25, and Database components.

The process begins when the user enters a query, which the system analyzes to understand its structure. The analyzed query is passed to the NLTTA model, where it is translated into the appropriate language for retrieval. The translated query is then sent to the BM25 model, which performs ranking and indexing of documents based on their relevance scores.

At this stage, a checkpoint is introduced: the system evaluates whether the score is greater than zero or not, indicating that at least one relevant document has been found. If the score is not zero, meaning relevant results exist, BM25 sends the indices of the top N documents to the system. The system then requests these documents from the database, which retrieves and returns them. The system displays the results to the user.

If the score equals zero, meaning no relevant documents match the query, the system displays a “No relevant news found” message and terminates the process. This explicit handling allows the system to gracefully terminate the process and inform the user by displaying “No relevant news found”. This improves the robustness and completeness of the model by ensuring that all possible outcomes of the ranking step are clearly addressed, preventing the system from proceeding with a request for documents that don't exist or returning an empty result without explanation.

CHAPTER 4: SYSTEM DESIGN

4.1 Design

System design is the process of representing architecture, interfaces, components that are included in the system. i.e., system design can be seen as the application of system theory to product development.

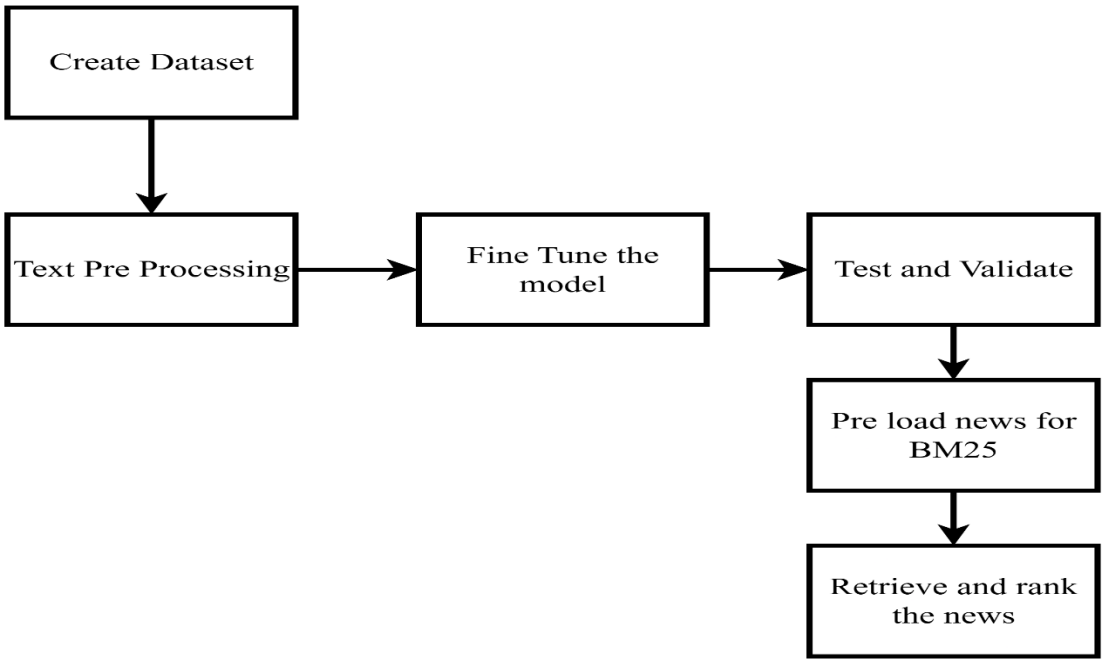

Figure 4.1: High Level Design of The System

The NLTTA system architecture is as shown in the Figure 4.1 above. The procedures involved are as follows:

- **Create Dataset**

  The foundational step of the entire system involves establishing a robust and comprehensive Dataset. The dataset was made by scrapping the Wikipedia chunks and refining the publicly distributed chunks by Wikipedia. Another dataset publicly cited by multiple papers was merged with our own and created the parallel corpus of Devanagari Nepali, Romanized Nepali and English, which is our final dataset

consisting of about 40,000. The dataset was spilt 70% for training (28000) and 20% (8000) and 10% (4000) for validation and test.

id: 0	english_sent	nepali_sent	roman_sent
0	It happened after the death of Saul, when Davi...	दाऊदले अमालेकीहरूलाई हराएर पछि सिकलग गए। यो शा...	dAUdale amAlekIharUlAI harAera paChi sikalaga ...
1	it happened on the third day, that behold, a m...	तब तेस्रो दिनमा एउटा जवान सैनिक सिकलगमा आयो। त...	taba tesro dinamA euTA javAna sainika sikalaga...
2	David said to him, "Where do you come from?" H...	दाऊदले त्यसलाई सोधे, "तिमी कहाँबाट आयौ?" त्यस ...	dAUdale tyasalAI sodhe, "timI kahA.NbATa Ayau?...
3	David said to him, "How did it go? Please tell...	दाऊदले भने, "मलाई भन, के भयो?" त्यसले भन्यो, "...	dAUdale bhane, "malAI bhana, ke bhayo?" tyasal...
4	David said to the young man who told him, "How...	दाऊदले त्यस सैनिकलाई भने, "तिमीले कसरी जान्यौ ...	dAUdale tyasa sainikalAI bhane, "timIle kasarI...
...	...	...	...

Figure 4.2: Sample of the parallel corpus used for training

- **Text Pre-processing**

  Once the dataset is secured, the raw textual data must undergo rigorous Text Pre Processing. This crucial stage is dedicated to cleaning the data, standardizing its format, and preparing it for machine consumption. The operations include tokenization, removing stop words, stemming or lemmatization, and overall noise removal including the html tags removal and normalization to ensure the model focuses only on relevant information.

  The Romanized Nepali sentences are added >>np_XX<< tags and the English sentences are added >>en_XX<< tags for the model to identify the language easily for translation tasks.

- **Fine Tune the Model**

  The system employs a sophisticated language model, called the NLTTA Model. In this step, a pre-trained version of mBART model is fine-tuned using the newly pre-processed dataset. This process is essentially specialized training, where the model's weights are adjusted and optimized to excel at translating the Romanized Nepali to English.

  The training loss and accuracy are examined after each epoch of training to detect and overfitting or underfitting by the model.

- **Test and Validate**

  After fine-tuning, the model's performance must be rigorously checked through the Test and Validate step. This involves evaluating the fine-tuned model against a separate, unseen test set of sentences to confirm that it meets the required performance metrics.

The test metrics used for NLTTA translation quality is BERT score and BLEU score. Successful validation ensures that the model is ready for deployment in a live operational environment.

- **Pre Load BM25**

  To enable efficient and rapid search capabilities, the collected news articles are pre-loaded and indexed for the BM25 component. BM25 is a specific and powerful algorithm used as a ranking function for document retrieval. This step involves creating an inverted index, which allows the BM25 algorithm to quickly calculate the relevance score of every document against a potential search query, preparing the entire news repository for fast retrieval.

- **Retrieve and Rank**

  The final step represents the operational goal achieved by the entire preparation process: the system's capability to Retrieve and rank the news. This is the culmination of the training effort, where the system is now capable of efficiently taking a user's query, utilizing the BM25 index to find the most relevant articles, and presenting them in an order of relevance, thus completing the search loop.

4.1.1 Refinement of Class Diagram

The refined class diagram is the detailed formed of the class diagram with the addition of precise definition of each attribute and operations of each class. The figure 4.1 below shows the refined class diagram of NLTTA. It consists of User class, which manages basic functionalities like entering a query and viewing results. The User initiates the process by creating a Query object, which holds the original and translated text and has methods for translation and validation. This Query is then handled by the APIService class, which serves as the primary interface for system operations, managing requests for both translation and search functionalities. The APIService calls upon specialized models and components, specifically utilizing the NLTTA Model for complex translation tasks, with the Translator acting as the intermediary.

The core search logic revolves around translation and ranking. The Translator class manages the flow, using methods like normalizeText() and translateToEnglish(). It uses the specialized NLTTA Model, defined by its modelPath and basic methods like loadModel() and

encode/decode(tokens), to perform the actual neural translation. Once translated, the query is processed by the BM25 class. This class stores crucial hyper-parameters like k_1 and b, and its primary function is to rankFeeds() by applying its scoring logic to the translated query.

Data storage and organization are managed by the Database, Feed, and FeedItem classes. The Database handles connections and queries via its connectionString and retrieval methods like getFeeds(). The database stores the Feed class, which organizes content based on attributes like language. Crucially, a single Feed contains multiple FeedItem objects. Each FeedItem represents an individual article, storing details like the title, content, and link, ensuring the BM25 algorithm has structured data to rank.

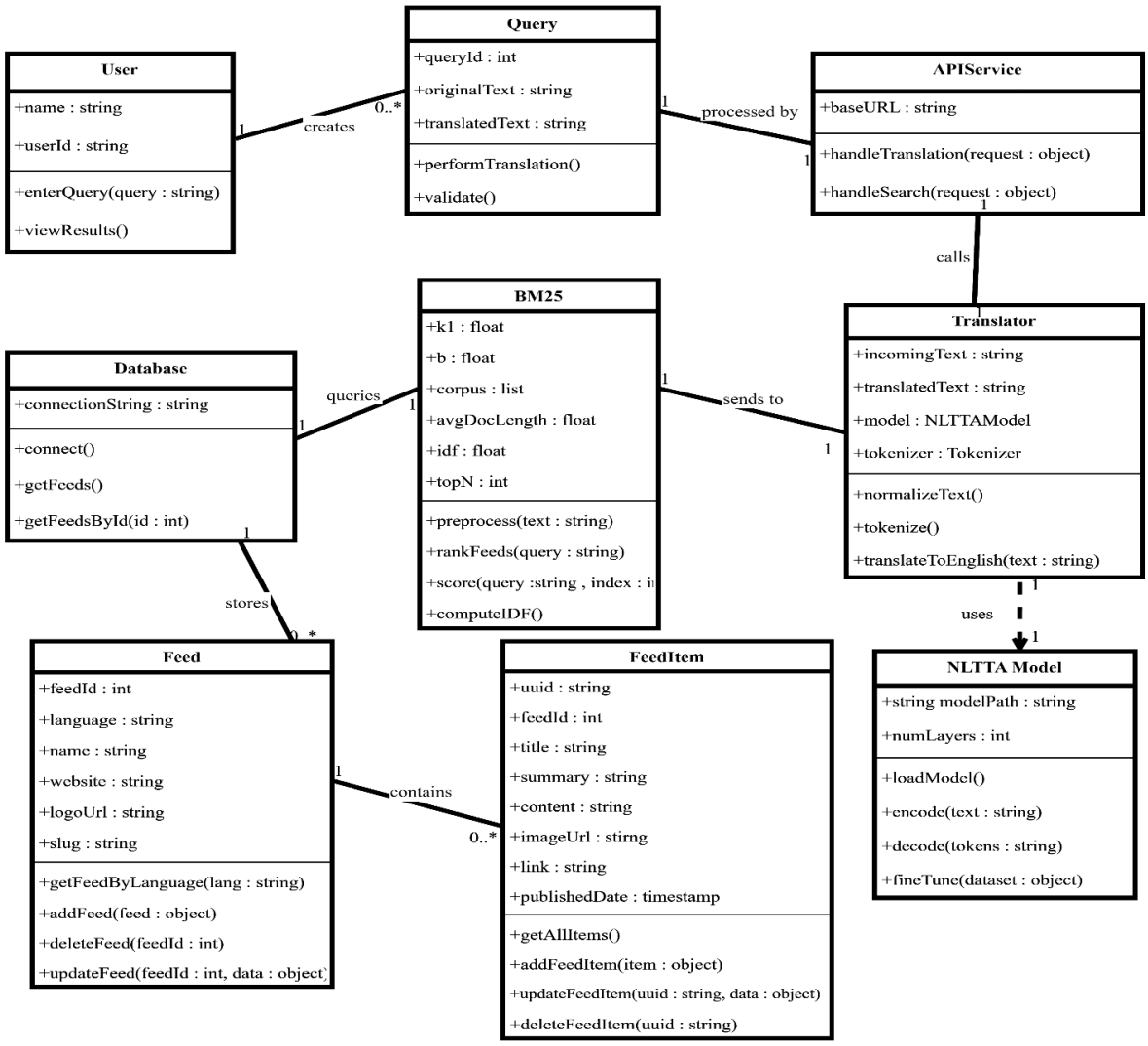

Figure 4.3 Refined Class Diagram of NLTTA System

4.1.2 Refinement of State Diagram

The refined state diagram is the detailed formed of the state diagram with the addition of precise definition of each attribute and operations of each class. The system's workflow begins

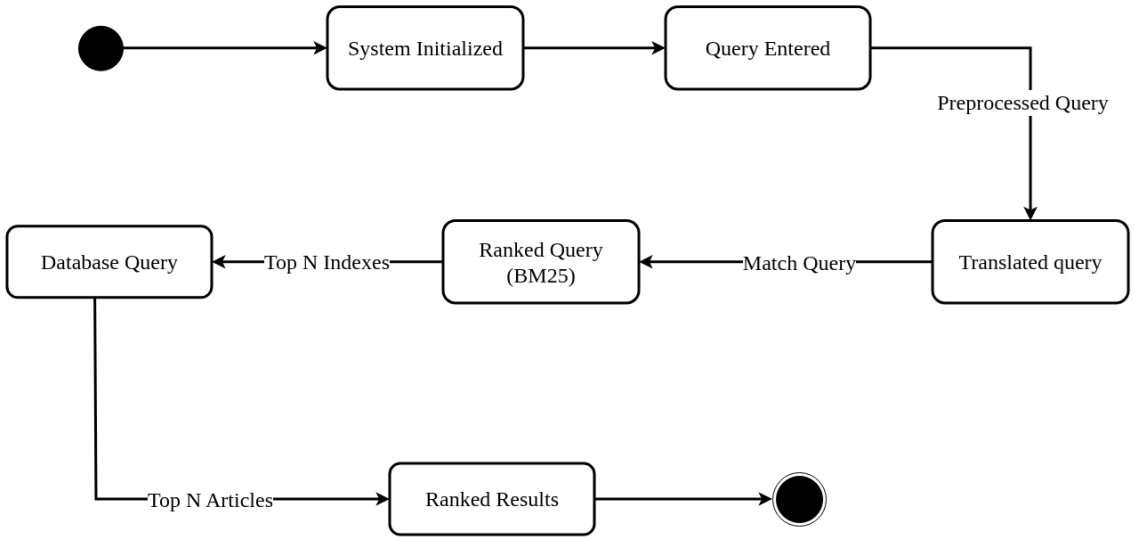

Figure 4.4 Refined State Diagram of The System Process

in the System Initialized state, signifying that all components are ready for operation. The process moves forward when a user submits an input, leading to the Query Entered state. From here, the query immediately transitions to the Preprocessed Query state, where it is cleaned and prepared for the translation step. This preprocessed input then leads to the Translated Query state, where the neural translation model converts the input into the language used for indexing. Once the translated query is ready, it is used to Match Query and proceed to the Ranked Query (BM25) state.

In the Ranked Query (BM25) state, the system employs the BM25 algorithm to calculate relevance scores and identify the best-matching articles, generating the Top N Indexes of those documents. These indexes are then used to trigger a Database Query to retrieve the actual content. The retrieved documents represent the Top N Articles state, which finally leads to the Ranked Results state, where the final, ordered list of articles is presented to the user, marking the end of the query lifecycle.

4.1.3 Refinement of Activity Diagram

The refined activity diagram is the detailed formed of the activity diagram with the addition of safety checks during the retrieval from database. The updated activity diagram shown below in Figure 4.3 represents the enhanced workflow of the query processing system, now including a check for the condition when the query is already in English.

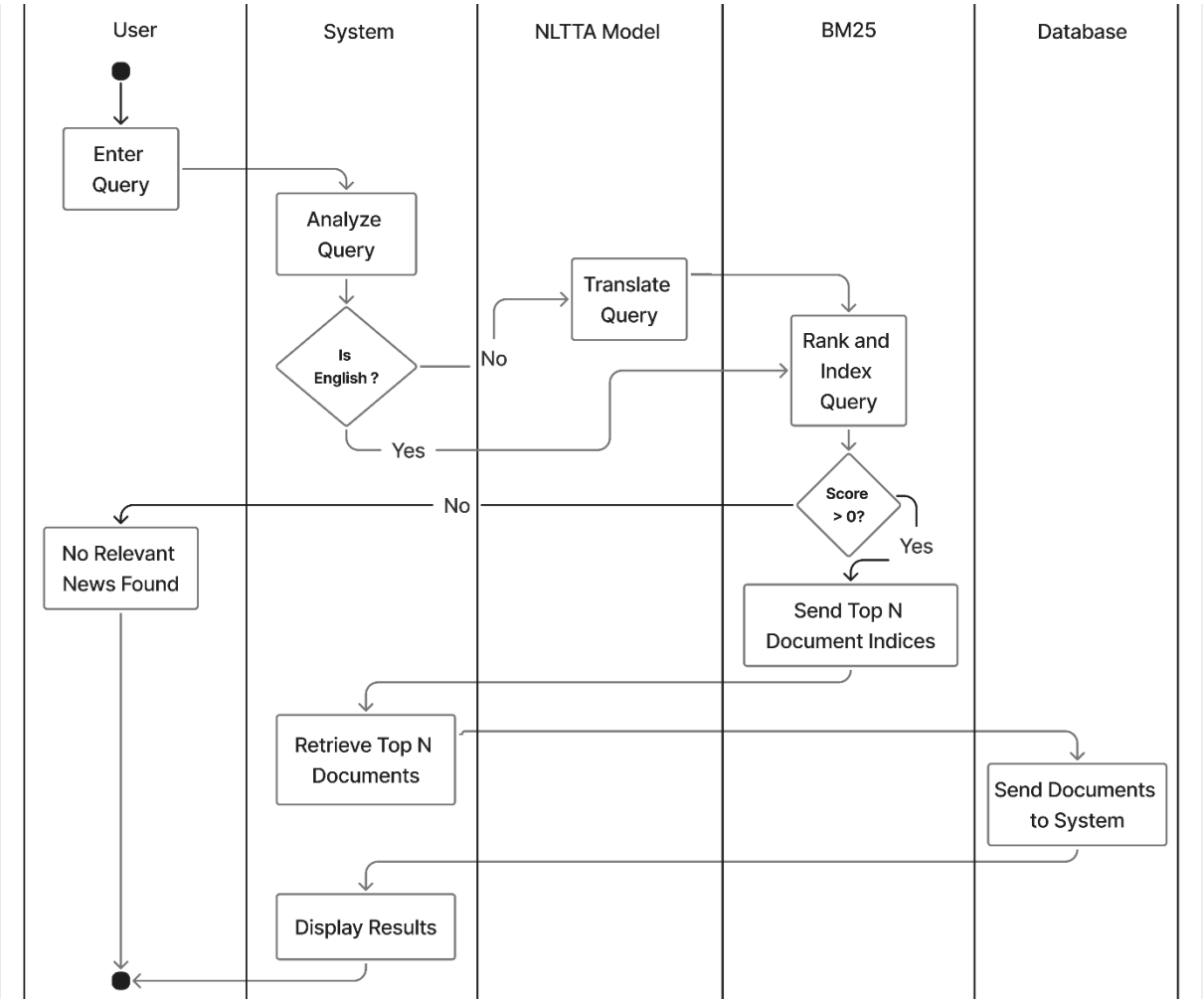

Figure 4.5 Refined Activity Diagram of Translation and Retrieval

The process begins when the user enters a query, which the system analyzes to understand its structure. At this stage, a checkpoint determines whether the query is in English. If it is already in English, the system bypasses the translation process and sends the query directly to the BM25 model for ranking and indexing, this saves computational resources and reduces response time. If the query is not in English, it is passed to the NLTTA model, where it is

translated into English before being sent to the BM25 model. The BM25 model then performs ranking and indexing of documents based on their relevance scores.

Another checkpoint is introduced at the BM25 stage to evaluate whether the score is greater than zero, indicating that at least one relevant document has been found. If the score is not zero, meaning relevant results exist, BM25 sends the indices of the top N documents to the system. The system then requests these documents from the database, which retrieves and returns them. The system displays the results to the user.

If the score equals zero, meaning no relevant documents match the query, the system displays a “No relevant news found” message and terminates the process. This explicit handling allows the system to gracefully end the process while informing the user of the outcome. This enhancement not only ensures robust handling of different cases but also improves efficiency by skipping unnecessary translation and preventing wasted computational effort, ultimately improving the system’s responsiveness and user experience.

4.1.4 Component Diagram

A component diagram is a type of UML diagram that shows the structural relationships and dependencies between the components of a software system. It illustrates how software components are connected and interact with each other within a system.

The process is centrally managed by the Backend and APIs component, labeled as the «Web Server», which acts as the system's primary orchestrator. It initiates the workflow by accepting the query from the front end NotifyNepal («Web App») and requires services from three core components to process it: NLTTA, BM25, and Database. The Backend requires the translated text of Romanized Nepali of the query which it gets from the get_translated_text service from NLTTA («Translataion»). Then the system calculates the scores and rank of the news with respect to the user query using score_and_rank service from BM25 («Rank»). Finally, it uses the ranked result to get the top relevant news from get_top_k service provided by the Database («Retrieval») to retrieve the final news content, before ultimately providing the get_relevant_news back to the user interface.

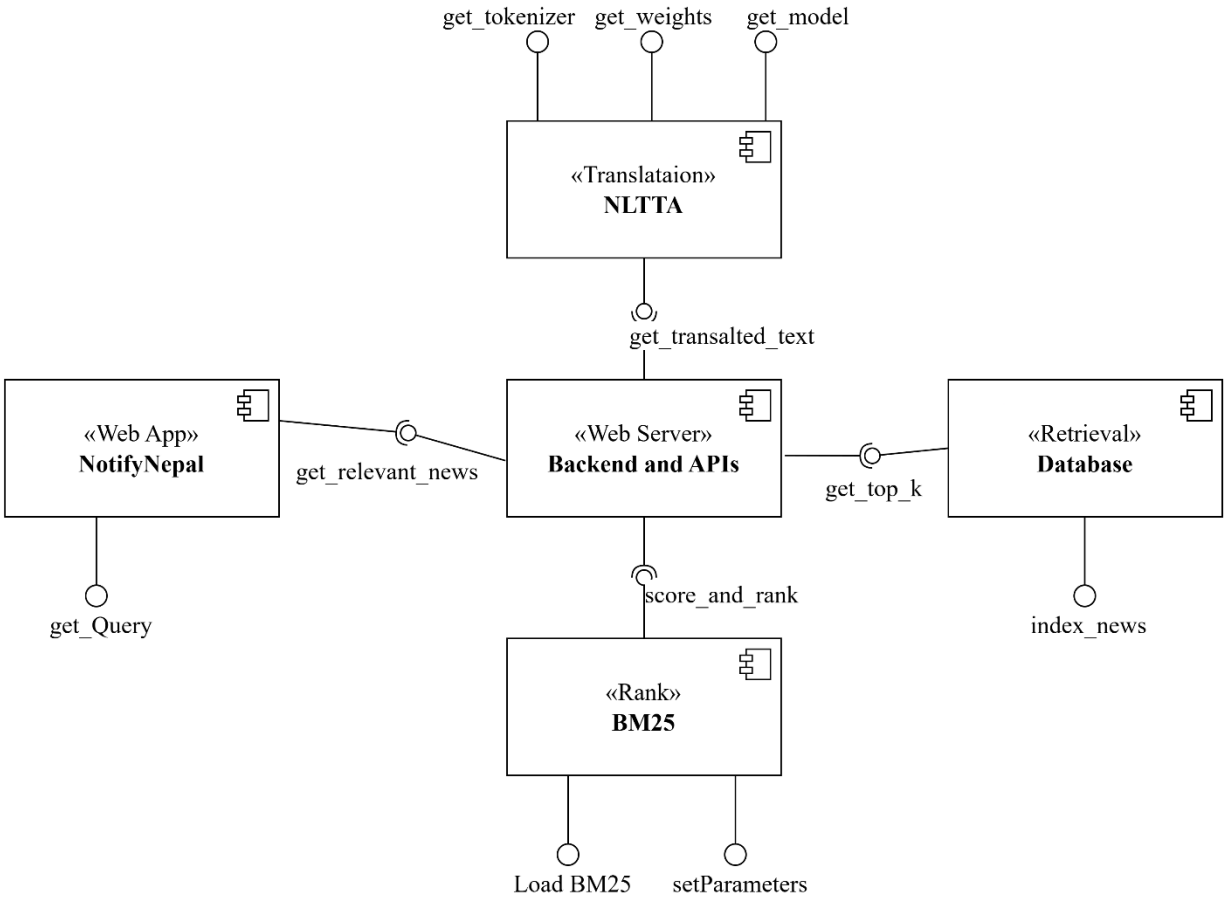

Figure 4.6: Component Diagram of The NLTTA System

The structure of the supporting components highlights a separation of concerns and component reusability. The NLTTA and BM25 components are designed with specific internal interfaces (get_tokenizer, setParameters) that suggest they are configurable machine learning and algorithmic models respectively, independent of the main application logic. This architectural choice, using required and provided interfaces (sockets and lollipops), ensures that the components can be easily replaced or updated without affecting the Backend's core logic, promoting system maintainability and scalability.

4.1.5 Deployment Diagram

The given diagram is a deployment diagram, which are UML structural diagrams that shows the relationships between the hardware and software components in the system and the physical distribution of the processing.

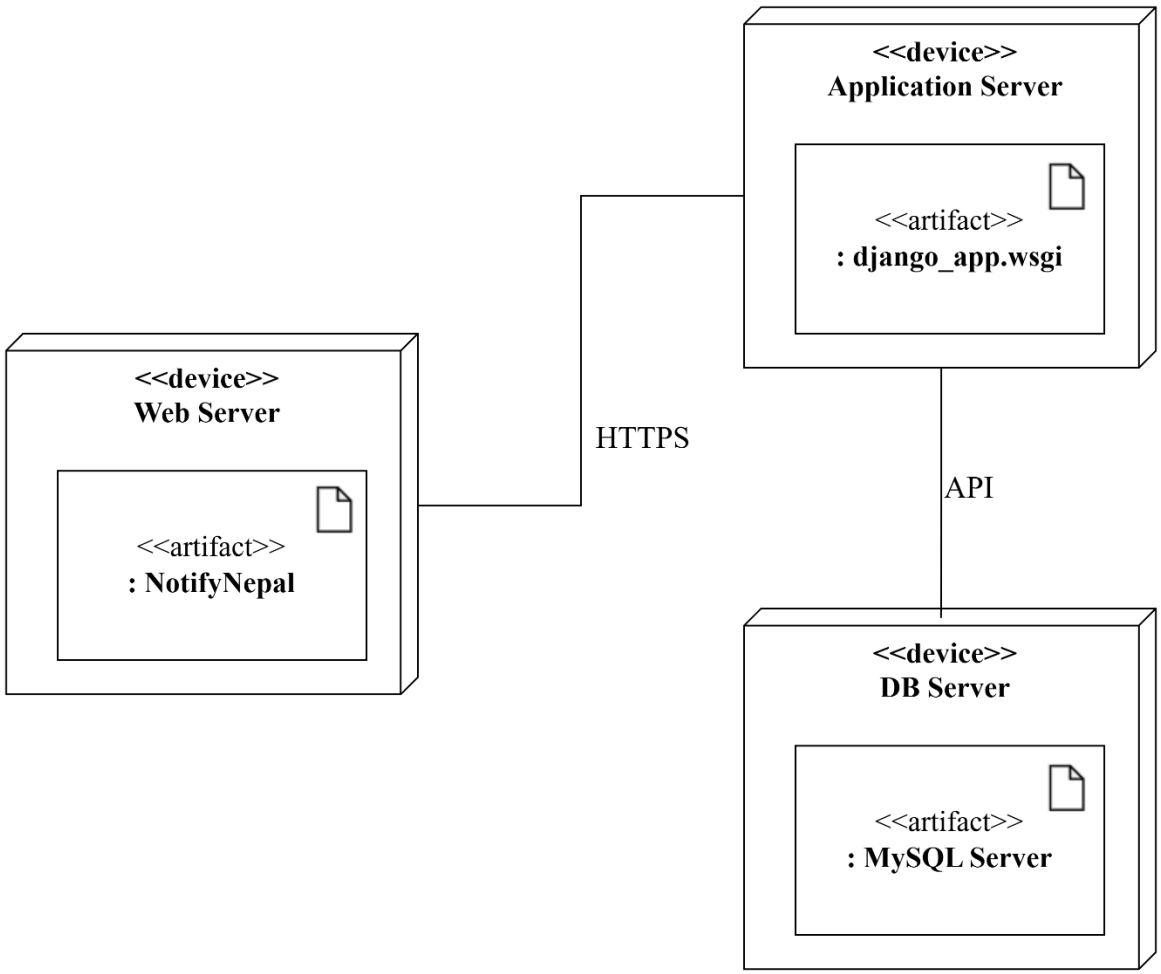

Figure 4.7: Deployment Diagram of the System

The deployment diagram illustrates the three-tier architecture of the system, showing how different components are deployed across various servers. The Web Server hosts the frontend artifact NotifyNepal, which provides the user interface and handles client-side interactions. It communicates securely with the Application Server over HTTPS. The Application Server contains the backend artifact django_app.wsgi, representing the Django based web application responsible for processing requests, executing business logic, and managing communication between the frontend and the database. The Application Server connects to the DB Server through an API to access or update data stored in the MySQL Server artifact. Overall, this deployment structure ensures modularity, security, and efficient communication between the web, application, and database layers of the system.

4.2 Algorithm Details

- **Translation Algorithm:**

The project employs a translation algorithm that uses a Neural Translation Approach. Specifically, the methodology involves fine-tuning on the domain of Romanized Nepali to English translation, utilizing the mBART model, which is based on the BART architecture. This approach leverages the power of pre-trained, large neural networks to adapt to a specific translation task with targeted domain data.

The steps of fine-tuning of NLTTA are detailed below:

1. Preprocess and Prepare the Data: This initial step involves data Normalization and the critical inclusion of special tokens to designate the source and target languages. For instance, the token >>ne_XX<< might be added to source language (like Nepali) sentences written in the Roman script, and >>en_XX<< would be used for English sentences. This helps the multilingual model understand the desired translation direction.

2. Load Resources and Set Languages: The pre-trained tokenizer and mBART model are loaded from open-source resources. Then, the source and target languages within the model's configuration are set to the special language tokens (e.g., >>ne_XX<< and >>en_XX<<) defined in the preprocessing stage.
3. Create Data Collator: A Data Collator is created to efficiently process the preprocessed dataset for training the Sequence-to-Sequence (Seq2Seq) model. This component is responsible for tasks like dynamic padding to create uniform batches.
4. Configure Seq2Seq Trainer: The Seq2Seq Trainer is configured with all necessary and suitable training parameters. These include, but aren't limited to, the loaded model, the learning rate, the total number of epochs (training cycles), and the output directory where results will be stored.
5. Train the Model: The core fine-tuning takes place in this step, where the preprocessed dataset is fed to the configured Seq2Seq Trainer to update the model's weights and adapt it to the specific translation domain. The parallel corpus used has over 40,000

pairs of Romanized Nepali to English. The model is then trained on this corpus for a set epoch of 40.

6. Evaluate and Save: Finally, the performance of the fine-tuned model is evaluated using a validation loss and training loss. Then complex test like BERTScore and BLEUScore are performed to get in-depth analysis on the model's translation quality. Upon satisfactory results, the trained model is saved for future use and deployment.

- **Retrieval Algorithm:**

The retrieval system employs a retrieval algorithm based on a Probabilistic Retrieval Approach. The specific Method used is term frequency saturation and document length normalization, which is implemented using the BM25 Model. This model is a ranking function used to estimate the relevance of documents to a given search query by considering the frequency of terms and adjusting for document length.

The BM25 algorithm is a highly effective ranking function that significantly refines the traditional TF-IDF model by introducing two adjustable parameters: k_1 and b. The parameter 'k_1' controls Term Frequency Saturation, ensuring that after a term is mentioned a few times, further mentions in the same document yield rapidly diminishing returns on the overall score. The parameter 'b' controls Document Length Normalization, which scales the document's term frequency based on its length relative to the average document length in the corpus, preventing long documents that mention the search terms once or twice from being unfairly ranked higher than shorter, more relevant documents. These modifications collectively make BM25 a superior ranking model, leading to more accurate relevance scores and better search results in practice.

The steps of algorithm are detailed below:

1. Indexing (Pre-computation):

- For every term 't' in the corpus of 'N' documents, compute its Document Frequency, df(t) (the number of documents containing 't').
- Compute the Average Document Length (avgDL) of the entire corpus. This is typically the sum of all document lengths divided by the total number of documents, N:

$$\text{avgDL} = \frac{1}{N}\sum_{D} |D|$$

2. Compute IDF for Each Term:

- Calculate the Inverse Document Frequency (IDF) for each term 't' using the formula:

$$\text{IDF(t)} = \log\left(\frac{N - df(t) + 0.5}{df(t) + 0.5} + 1\right)$$

3. Set Hyper-parameters:

- Determine the values for the two core parameters.
  - k_1: Controls Term-Frequency Saturation (how quickly the term's score contribution diminishes as its frequency in a single document increases). A common range is $1.2 \leq k_1 \leq 2.0$. The NLTTA's BM25 sets k_1 as 1.5.
  - b: Controls Document-Length Normalization (how much the document's length, relative to avgDL, penalizes the term's score). A common value is b=0.75. The NLTTA's BM25 sets b as 0.8.

4. Compute Term's BM25 Contribution:

- For each query 'Q' and for each candidate document 'D':
  1. For each query term $q \in Q$, get its term frequency in the document f(q,D).
  2. Compute the term's individual BM25 contribution (Term Weighting and Normalization Component) using the formula:

$$\text{Score}_q(D) = \text{IDF}(q) * \frac{f(q,D)(k_1+1)}{(f(q,D) + k_1\,(1 - b + b * \frac{|D|}{\text{avgDL}}}$$

The denominator includes the saturation component and the length normalization component $\frac{|D|}{\text{avgDL}}$ is the document length ratio).

- Aggregate Term Scores: The final BM25 score for the document 'D' with respect to the query 'Q' is the sum of the individual contributions of all terms present in the query:

$$\text{BM25(D, Q)} = \sum_{q \in Q} \text{score}_{q(D)}$$

5. Rank Documents:

a. Sort all candidate documents by their BM25 scores in descending order.
b. Return the top-k highest-scoring documents as the retrieval result.

CHAPTER 5: IMPLEMENTATION AND TESTING

5.1. Implementation Overview

Process model used:

As an Agile methodology, NLTTA development process emphasizes on collaboration between developers, stakeholders, and customers. The development team works closely with the users and market to identify their needs and requirements and deliver working software quickly and frequently. This allows the users and other collaborators to see progress and provide feedback along the way, which can be incorporated into the next iteration.

5.1.1 Tools Used

5.1.1.1 Front End Tools

The front-end of the NLTTA project was built using NextJS, which is a popular framework for building application. Next.js is renowned because it is a React framework that provides a robust, production-ready structure for building fast, full-stack web applications with built-in features like SSR and SSG. With addition to NextJS, Tailwind CSS was used for the basic styling and structure of the webpage.

5.1.1.2 Back End Tools

The back-end of the NLTTA project was built using Django, which is a popular Python web framework for building APIs and web applications. Django is one of the most popular Python web-framework known for its emphasis on rapid development and clean, pragmatic design, allowing developers to build complex, database-driven websites quickly. With addition to Django, NLTK was used for preprocessing the dataset, PyTorch and Transformers libraries were used to fine tune the mBART model, Pandas and Numpy to load and store the data and to read the datasets and BeautifulSoup for scraping the news articles. Finally, MySQL was used for database management for its simple and wide support throughout python frameworks.

5.1.1.3 QA Testing Tools

The performance test of NLTTA was done using different tools for different uses. Postman was used to test the APIs developed in the backend. Similarly, BERT score and BLEU score were used to measure the translation model.

5.1.2 Implementation Details of Modules

- **Text Cleaning and Preprocessing module:**

This module is crucial for preparing raw text data for analysis in Natural Language Processing (NLP). It converts text to a normalized form for tokenization and thus allowing translation and retrieval tasks efficiently.

```
def lemmatize(self, text):
        tokens = word_tokenize(text)
        tagged = pos_tag(tokens)
        lemmatized = [
            self.lemmatizer.lemmatize(word, self.get_wordnet_pos(tag))
            for word, tag in tagged
        ]
        return ' '.join(lemmatized)


    def remove_html(self, text):
        soup = BeautifulSoup(text, "html.parser")
        return soup.get_text(separator=' ', strip=True)


    def stop_word_removal(self, text):
        filtered = [word for word in text.split() if word not in stop_words]
        return ' '.join(filtered)
```

```
def pre_process(self, text):
    text = self.remove_html(text)
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r'\s+', ' ', text).strip()
    text = self.stop_word_removal(text)
    text = self.lemmatize(text)
    return text

def tokenize(self, text):
    return text.lower().split()
```

The core method, pre_process, performs a series of transformations. First, it uses BeautifulSoup in remove_html to strip all HTML tags, then it converts the text to lowercase for uniformity. Next, it uses regular expressions to remove punctuation and normalize whitespace. Following that, it calls stop_word_removal to eliminate common, less meaningful words like 'a' and finally, it applies lemmatization in the lemmatize function to reduce inflected words to their root form (lemma) using part-of-speech tagging. This entire sequence transforms unstructured, noisy input text into a clean, normalized format suitable for translation and retrieval.

```
Before Cleaning: <p>Prime Minister is in Pokhara.<p> <div>The President visited Kathmandu today.</div>
 <span>Tourists love the city of Bhaktapur!</span> <b>New airport opened in Lumbini.</b>
After Cleaning: prime minister pokhara president kathmandu tourists love city bhaktapur new airport lumbini
```

Figure 5.1: Text Before and After passing through the Cleaning Module

- **Retrieval Module**

The retrieval module is responsible to create indexed dictionary of all news available in the database once at the start of the application deployment. Then for each query the module is responsible to find the most relevant news and provide the result to the user.

```
def compute_doc_freqs(self):
        df = defaultdict(int)
        for doc in self.corpus:
            for term in set(doc):
                df[term] += 1
        return df


    def compute_idf(self):
        idf = {}
        N = len(self.corpus)
        for term, freq in self.df.items():
            idf[term] = math.log(1 + (N - freq + 0.5) / (freq + 0.5))
        return idf


    def score(self, query, index):
        doc = self.corpus[index]
        doc_len = self.doc_lens[index]
        tf = Counter(doc)
        score = 0.0
```

```
        for term in self.tokenize(self.pre_process(query)):
            if term in tf:
                term_freq = tf[term]
                term_idf = self.idf.get(term, 0)
                denom = term_freq + self.k1 * (1 - self.b + self.b * doc_len / self.avgdl)
                score += term_idf * (term_freq * (self.k1 + 1)) / denom
        return score

    def rank(self, query, top_k=5):
        scores = [(i, self.score(query, i)) for i in range(len(self.corpus))]
        scores.sort(key=lambda x: x[1], reverse=True)
        return scores[:top_k] if top_k else scores
```

This retrieval module is designed for document ranking based on a query, utilizing techniques from Information Retrieval, specifically an implementation of the BM25 scoring algorithm. The process begins with two core methods: compute_doc_freqs and compute_idf. The compute_doc_freqs method iterates through the document corpus to calculate the document frequency of each unique term (how many documents contain the term), storing these counts in a 'defaultdict'. Building on this, the compute_idf method calculates the IDF for every term, using a formula that incorporates the total number of documents (N) and the term's document frequency. This weighting scheme ensures that rare terms are given higher importance in the final ranking score.

The actual retrieval and ranking are handled by the score and rank methods. The score(self, query, index) method calculates the BM25 score between a pre-processed query and a specific document identified by its index. This calculation combines the term's pre-computed IDF with the term's TF within the document, adjusted by the document length relative to the average document length using the BM25 parameters k_1 and b in the 'denom' calculation. Finally, the

rank(self, query, top_k=5) method executes the full ranking process by iterating through the entire corpus, calculating the BM25 score for each document using the score method, and then returning the 'top-k' documents (default to 5) based on their highest scores.

```
prime minister
Index:  192   Score:  7.73181744792945
Title:  Prime Minister Oli to address House of Representatives on current issues
link:  https://english.onlinekhabar.com/prime-minister-oli-parliament.html

Index:  636   Score:  7.686156030211757
Title:  PM Oli holds bilateral meetings with Egyptian and Portuguese counterparts
link:  https://nepalnews.com/s/diplomacy/pm-oli-holds-bilateral-meetings-with-egyp

Index:  409   Score:  7.6148375577007155
Title:  PM Oli returning home after attending UN Conference in Spain
link:  https://nepalnews.com/s/diplomacy/pm-oli-arriving-home-today/
```

Figure 5.2: Retrieval Module Output

- **Translation Module**

The translation module is responsible to translate Romanized Nepali queries from users to English after they undergo the pre-processing process and before the retrieval process. Its primary function is to perform NMT, specifically translating a source text, assumed to be in Nepali (ne_XX), into English (en_XX).

```
def create(self, request, *args, **kwargs):
    try:
        source_text = request.data.get('source_text')
        if not source_text:
            return Response({"error": "source_text is required"}, status=status.HTTP_400_BAD_REQUEST)

        text = source_text
        inputs = tokenizer(self.pre_process(text), return_tensors="pt")
```

```
            generated = model.generate(
                **inputs,
                forced_bos_token_id=tokenizer.lang_code_to_id["en_XX"],
                max_length=128,
                num_beams=5
            )
            translated_text = self.remove_word_prefixes(str(tokenizer.decode(generated[0], skip_special_tokens=True)), ['e', 'ne'])


            serializer = self.get_serializer(data=request.data)
            serializer.is_valid(raise_exception=True)
            serializer.save(translated_text=translated_text)


            return Response(serializer.data, status=status.HTTP_201_CREATED)
        except Exception as e:
            return Response({
                "error": str(e),
                "trace": traceback.format_exc()
            }, status=500)
```

The process begins by retrieving the source_text from the incoming request. It then initializes a pre-trained MBart50 model and its corresponding tokenizer, loading weights from a specified checkpoint. The input text is pre-processed (using a presumed self.pre_process method) and tokenized. The core translation occurs using the model.generate() method, which applies the sequence-to-sequence model with constraints like a forced English target language token

(forced_bos_token_id), a maximum length, and beam search (num_beams=5) to generate the highest-probability English translation.

Following generation, the raw output is decoded back into human readable text using tokenizer.decode(), and a custom self.remove_word_prefixes method cleans up any residual artifacts from the tokenization process. Finally, the module handles the API response and data persistence. It uses a serializer to validate the original request data and saves the newly generated translated_text to the database, as indicated by serializer.save(). The entire operation is wrapped in an exception handler to gracefully catch and return any errors that occur during the translation or processing, ensuring the API returns a structured response with the translated text upon success.

```
pardhan mantri
<Response status_code=201, "text/html; charset=utf-8">
prime minister
Index:  192   Score:  7.73181744792945
Title:  Prime Minister Oli to address House of Represer
link:  https://english.onlinekhabar.com/prime-minister-

Index:  636   Score:  7.686156030211757
Title:  PM Oli holds bilateral meetings with Egyptian a
link:  https://nepalnews.com/s/diplomacy/pm-oli-holds-b
```

Figure 5.3: Translation Module Output for "pardhan mantri"

- **Transliteration Module**

The transliteration module is based on Indic-Transliteration which is a rule-based transliteration for Devanagari. It reads data from an Excel file containing Nepali sentences, performs transliteration into ITRANS format, and saves the output to a new Excel file for easy reference or further processing. This module helps bridge linguistic and technical gaps by allowing Nepali text to be processed, searched, or analyzed in Roman script making it useful for natural language processing, data preprocessing, and multilingual applications.

```
class Transliteration:

    def __init__(self, filepath):
```

```
    self.filepath = filepath
    self.df = None

def load_data(self):
    self.df = pd.read_excel(self.filepath)
    print("Data loaded successfully")
    print(self.df.head())

def transliterate_nepali(self, source_col='nepali_sent', target_col='roman_sent'):
    if self.df is None:
        raise ValueError("Data not loaded")
    roman_sent = []
    for row in self.df[source_col].astype(str):
        roman_nepali = transliterate(row, DEVANAGARI, ITRANS)
        roman_sent.append(roman_nepali)
    self.df[target_col] = roman_sent
    print("Transliteration completed: ")
    print(self.df.head())

def save_to_excel(self, output_path='transliterated_output.xlsx'):
    if self.df is not None:
        self.df.to_excel(output_path, index=False)
        print(f"File saved successfully to '{output_path}'.")
```

```
    else:
        print("No data to save. Please run transliteration first.")
```

The TransliterationModule takes Nepali sentences written in the Devanagari script from an Excel file, processes them, and converts each sentence into its Romanized equivalent using the ITRANS standard. This transformation enables easier reading, linguistic analysis, and text processing in systems that do not natively support Devanagari. Once the transliteration process is complete, the module prints the first few rows of the dataset, showing both the original Nepali sentences and their corresponding Romanized forms to verify that the operation has been successfully performed.

```
Transliteration completed:
      nepali_sent                          roman_sent
0       नेपाल एक सुन्दर देश हो।          nepAla eka sundara desha ho|
1     प्रधानमन्त्री कठमाडौंमा छन्।     pradhAnamantrI kAThmANDaumA chhan|
2  लुम्बिनी बुद्धको जन्मस्थल हो।       lumbinI buddhako janmasthala ho|
3      आज मौसम रमाइलो छ।                Aja mausama rAmro cha|
4   विद्यार्थीहरू विद्यालय गए।          vidyArthIharU vidyAlaya gaye|

File saved successfully to 'transliterated_output.xlsx'.
```

Figure 5.4: Transliteration Module Output

- **News Conversion Module**

The news conversion module is used to convert the Devanagari Nepali news retrieved from APIs to English for storing in Database and searching during the retrieval. This module converts the Devanagari Nepali news to English using the default mBART model without the >>ne_xx<< tag.

```
def translated_news(nep_news_json="nep_news.json"):
    with open(nep_news_json, "r", encoding="utf-8") as f:
        data = json.load(f)
    translated_list = []
```

```
for item in data:
    item["content"] = remove_html(item["content"])
    item["title"] = remove_html(item["title"])
    print(item["title"])

    text = item["title"] + " " + item["content"]
    inputs = tokenizer(
        remove_html(text),
        return_tensors="pt",
        truncation=True,
        padding="max_length",
        max_length=1024
    ).to(device)

    generated_tokens = model.generate(**inputs,
        forced_bos_token_id=tokenizer.lang_code_to_id["en_XX"])

    translation = tokenizer.batch_decode(generated_tokens, skip_special_tokens=True)[0]

    translated_list.append({
        "title": item["title"],
        "translation": translation
    })
```

```
print("Feed items imported successfully.")
```

The module implements a method dedicated to batch translation of news articles from Nepali into English, following a standard machine translation pipeline. It begins by loading the articles from a JSON file (nep_news.json). For each article, it first cleans the title and content by removing HTML tags, then combines them into a single string. This text is tokenized using a model-specific tokenizer with parameters like max_length=1024 and padding to prepare it for the neural model. The core translation is performed by the pre-trained MBart model using model.generate(), explicitly setting the target language to English (en_XX). Finally, the generated tokens are decoded back into a human-readable English translation, which is then stored along with the original title in a list for eventual output. This method is essentially a dedicated script for mass-converting news data between these two specific languages.

```
Before:  प्रधानमन्त्री पोखरामा छन्।
After:  Prime Minister is in Pokhara today. The Prime Minister inaugurated a
Before:  राष्ट्रपतिले काठमाडौंमा सम्बोधन गर्नुभयो।
After:  The President addressed the Conference in Kathmandu. The President s
Before:  पर्यटकहरूले भक्तपुरको भ्रमण गरे।
After:  Tourists visited Bhaktapur. The cultural heritage of Bhaktapur is at
Before:  लुम्बिनीमा नयाँ विमानस्थल सञ्चालनमा आयो।
After:  A new airport was launched at the airport in Odisha. The airport has
Before:  काठमाडौंमा प्रदूषण बढ्दै गएको छ।
After:  Pollution is rising in the city of Kathmandu, and air quality is dec
Feed items imported successfully.
```

Figure 5.5: News Conversion Module

5.2 Testing

5.2.1 Test Cases of Unit Testing

Unit testing involved testing individual components or modules of the application system to ensure that each module functions correctly in an isolated environment. All of the modules were tested independently using test cases designed to verify both valid and invalid inputs.

- **Translation Module Testing**

The unit testing of Translation module is solely focused on testing the translation unit under different circumstances like basic word translation to ambiguous word and phrase translations.

Table 5.1 Test cases for Translation Unit (NLTTA)

Test Case ID	Description	Input (Roman Nepali)	Expected Output (English)	Actual Output	Status	Remarks
UT_NLTTA_001	Translate a simple greeting	"Namaskara"	"Greetings"	"Greetings"	Pass	Basic word translation
UT_NLTTA_002	Translate a common phrase	"Timi kasto chau?"	"How are you?"	"How are you?"	Pass	Simple sentence with question
UT_NLTTA_003	Translate with mixed vocabulary	"ma school janchu "	"I go to school"	"I go to school"	Pass	Correct phrase with verb tense
UT_NLTTA_004	Translate unknown/ambiguous words	"Chaaina"	"No"	"No"	Pass	Check handling of ambiguous words

The Table 5.1 above shows the test cases for translation validation. The test cases focus on basic translation competency, covering simple greetings (UT_NLTTA _001), common phrases (UT_NLTTA _002), correct sentence structure (UT_NLTTA _003), and ambiguous wording (UT_NLTTA _004) all of which successfully achieved a "Pass" status with the actual output matching the expected English translation.

The test was conducted on the translation module, showing the output:

```
[04/Nov/2025 20:56:17] "GET /api/feed-items/search?q=Timi kasto chau HTTP/1.1" 301 0
Received:  Timi kasto chau
Translated:  how are you

[04/Nov/2025 20:55:20] "GET /api/feed-items/search?q=Namaskara HTTP/1.1" 301 0
Received:  Namaskara
Translated:  greetings
```

Figure 5.6: Result for NLTTA Test Case 1 and 2

```
[04/Nov/2025 20:54:14] "GET /api/feed-items/search?q=Chaaina HTTP/1.1" 301 0
Received:  Chaaina
Translated:  no

[04/Nov/2025 20:57:49] "GET /api/feed-items/search?q=ma school janchu HTTP/1.1" 301 0
Received:  ma school janchu
Translated:  i go to school
```

Figure 5.7: Result for NLTTA Test Case 3 and 4

- **Retrieval Module Testing**

This model manages the retrieval of most relevant documents after translation. The test of this model included queries of mixed Romanized Nepali and English for retrieval and testing the IDF calculation of the BM25 model.

Table 5.2: Test cases for Retrieval Unit (BM25)

Test Case ID	Description	Input Query (English)	Expected Output	Actual Output	Status	Remarks
UT_BM25_001	Compute IDF correctly	"student"	IDF > 0 for rare terms	IDF = 3.15	Pass	Verifies correct IDF computation
UT_BM25_002	Ranking order accuracy	Query: "vidharthi scholarship"	Top 5 docs contain "student"	Ranked correctly	Pass	Confirms retrieval relevance

Test Case ID	Description	Input Query (English)	Expected Output	Actual Output	Status	Remarks
UT_BM25_003	Empty query handling	“”	Return “Invalid query”	Empty	Pass	Edge case handling
UT_BM25_004	Stopword only query	“the and a”	Return “No relevant news”	“No Result”	Pass	Checks stopword filtering

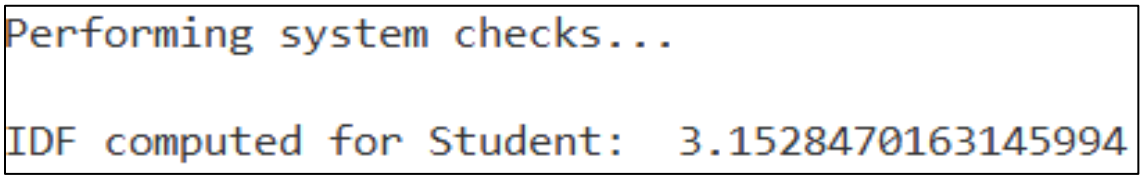

Figure 5.8: BM25 Test Case 1 Result

Figure 5.9: BM25 Test Case 3 and 4 Result

Figure 5.10: BM25 Test Case 2 Result

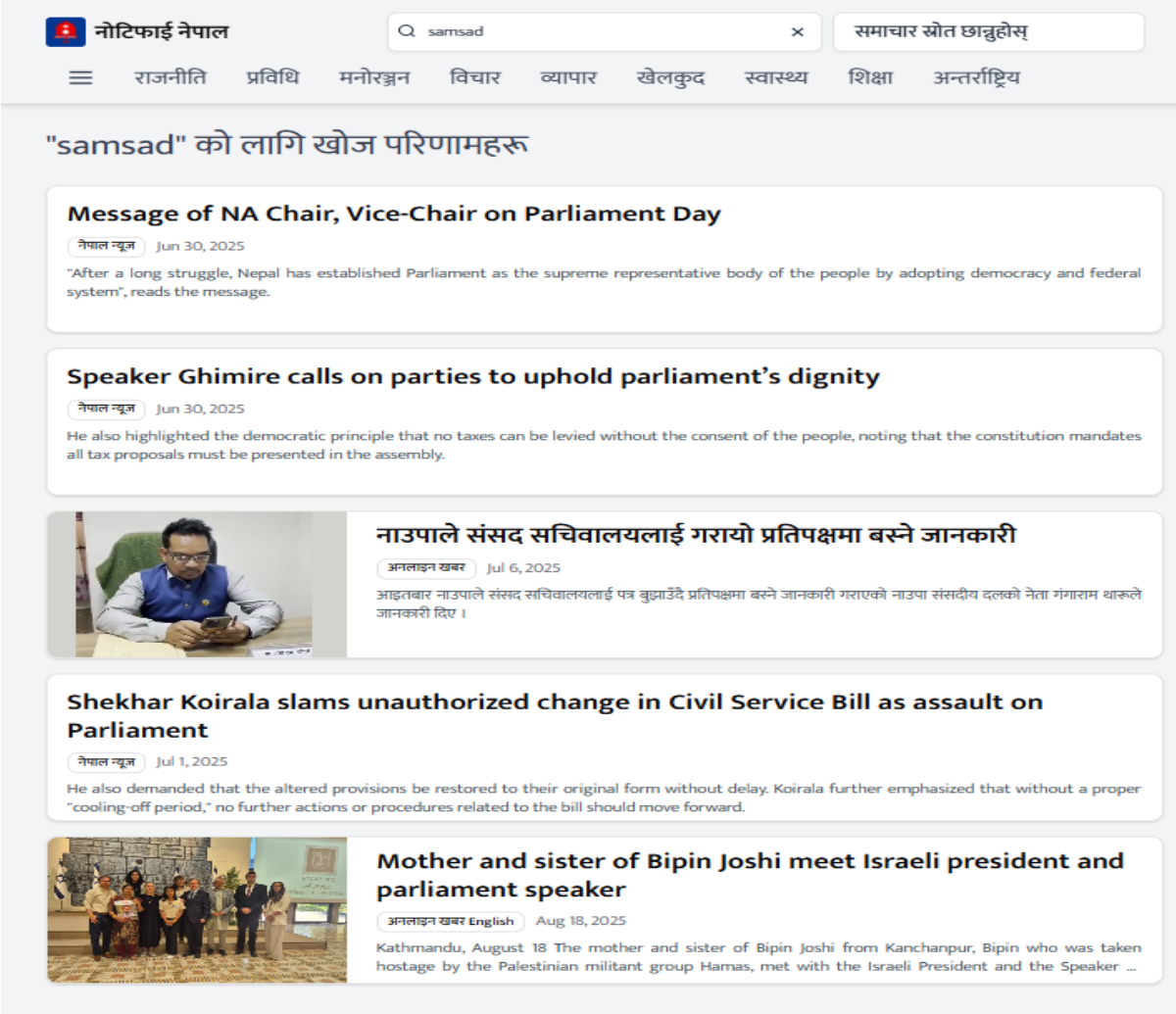

Figure 5.11: BM25 Test Case 5 Result

BLEU Score

BLEU score is a metric used to evaluate the quality of machine-translated text by comparing it to one or more reference translations. It measures the overlap of n-grams between the candidate and reference, with higher scores indicating better translation accuracy. BLEU relies on exact n-gram matches, which may penalize valid translations with different wording.

The BLEU score test was conducted using a list of Romanized Nepali inputs corresponding to their English references. The English and Romanized Nepali parallel corpus used for testing was a custom created unseen corpus with ambiguous and variable sentences to get the most accurate measurement as possible. The figure 5.5 below shows the performance varied vastly based on the nature of the sentences.

```
Corpus BLEU Score: 39.84

Sentence-level BLEU scores:
Sentence 1: 31.62
Sentence 2: 100.00
Sentence 3: 100.00
Sentence 4: 56.23
Sentence 5: 100.00
Sentence 6: 56.23
Sentence 7: 66.87
Sentence 8: 28.57
Sentence 9: 56.23
Sentence 10: 24.03
Sentence 11: 39.76
Sentence 12: 39.76
Sentence 13: 100.00
Sentence 14: 100.00
Sentence 15: 100.00
Sentence 16: 17.10
Sentence 17: 7.73
Sentence 18: 54.75
Sentence 19: 100.00
Sentence 20: 6.85
Sentence 21: 26.59
Sentence 22: 16.35
...
```

Figure 5.12: BLEU Score Test of NLTTA

The BLEU score ranges from 0-100, on the scale of how many n-grams matched for the translation task. The perfect translation score for BLEU score would need to be an exact same match (word to word). Even if the different order of words may convey the same meaning the BLEU score will take a hit. The BLEU score scale is generally categorized as:

Table 5.3 BLEU Score Range

BLEU Score Range	Quality Level	Interpretation and Remarks
0–10	Almost Useless	The translation is hard to understand and has virtually no similarity to a professional human translation.
10–20	Poor	The gist is difficult to get, and major editing or re-translation is likely required to make it useful.
20–29	Adequate	The gist is clear, but the output contains significant grammatical errors and awkward phrasing.
30–40	Understandable to Good	The translation is generally understandable and serviceable, though it may still contain minor issues or lack natural fluency.
40–60	Good to High Quality	The translation is understandable and fluent, requiring only minor editing for style or perfection. This range is often considered publishable quality.
>60	Very High Quality	Often indicates very high quality, but can also be a sign of a flawed metric (e.g., test set reuse) or overfitting to the specific reference translations.

The BLEU score of 39.84 for the NLTTA model signifies that its translations are of good, production-ready quality within the context of a specialized machine translation task. According to standard metrics, a score in the 30-40 range is classified as “Understandable to Good”, meaning the output translations are generally clear, accurately convey the core message, and are highly serviceable for practical use.

BERT Score

BERT score evaluates text similarity by comparing contextual embeddings of words using a pre-trained BERT model. It captures semantic meaning and aligns words based on their context rather than exact surface forms. BERT score measures deep semantic similarity, making it more robust to synonyms and paraphrasing.

The BERT score test was conducted using a list of roman Nepali inputs corresponding to their English references.

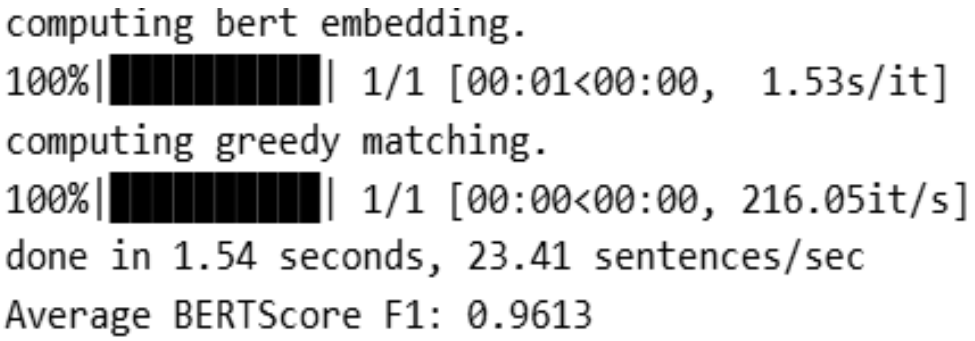

Figure 5.13: BERT Score of NLTTA

The BERTScore (F1) of 0.9613 achieved by the NLTTA model is an excellent indicator of the semantic quality of its translations, especially when considering the model's small scale. BERTScore uses contextual embeddings based on BERT to measure how closely the generated translation aligns in meaning with the reference translation. A score this high, near the maximum of 1.0, confirms that NTA's outputs are semantically accurate and carry the same core meaning as the desired English text.

5.2.2 Test Cases for System Testing

Table 5.4: Test Cases for NLTTA System Testing

Test Case ID	Description	Input	Expected Output	Actual Output	Status	Remarks
ST_001	Verify system initializes properly (frontend, API, DB, model)	Start application	All modules initialize successfully	All modules loaded	Pass	Checks model loading & DB connection
ST_002	Search query in Romanized Nepali	"vidharthi"	Returns English-translated query and relevant English news	Correct news list displayed	Pass	Confirms translation & BM25 retrieval link
ST_003	Search query in Devanagari Nepali	"विद्यार्थी"	Converts to English and retrieves related articles	Correct news list displayed	Pass	Tests transliteration → translation
ST_004	Search with mixed Nepali-English input	"Nepal ko prime minister"	Returns relevant results for "Prime Minister of Nepal"	Results shown correctly	Pass	Validates mixed-input handling
ST_005	No matching news article found	"jupiter ko student"	Displays "No relevant news found"	Message displayed	Pass	Tests error handling

Test Case ID	Description	Input	Expected Output	Actual Output	Status	Remarks
ST_006	System under poor network	Query: “sansad”	Should respond within 4 s	2.5s response	Pass	Checks performance constraint
ST_007	System response to concurrent users	6 simultaneous queries	Each returns correct result	All completed within 15s	Pass	Tests concurrency performance

System testing validated the end-to-end functionality of the NLTTA system. It confirmed that the translation, transliteration, and retrieval components work together seamlessly to deliver relevant news results from various input types. The tests also verified performance, concurrency handling, and proper user feedback for invalid or unmatched queries. Overall, the system met its functional and non-functional requirements effectively.

5.3 Result Analysis

5.3.1 Training Loss Analysis:

NLTTA was trained in multiple session using checkpoints. Each session containing anything from 7000 to 12500 itemset of the dataset. The figure 5.14 below shows the result of loss obtained during training of the second session containing 7000 items.

The training loss decreased progressively with an increase in training steps, demonstrating effective model learning and convergence. Initially, at 500 steps, the training loss was relatively high (1.1967), indicating that the model parameters were still in the early stages of adjustment. As the training continued, the loss showed a significant reduction, reaching 0.4578 by 2000 steps, which reflects rapid improvement in the model’s ability to fit the training data. After 4000 steps, the rate of loss reduction became slower, gradually stabilizing around 0.3449 at 7000 steps. This gradual flattening of the loss curve signifies that the model was approaching

convergence and had learned the key patterns in the dataset. Minor fluctuations, such as at 3000 steps, are normal and attributed to variations during stochastic optimization. Overall, the consistent decline in training loss confirms that the model was successfully trained, showing stable learning behavior and effective parameter optimization over time.

Step	Training Loss
500	1.196700
1000	0.509000
1500	0.467700
2000	0.457800
2500	0.407300
3000	0.415300
3500	0.391100
4000	0.379500
4500	0.384300
5000	0.370000
5500	0.362900
6000	0.362500
6500	0.360800
7000	0.344900

Figure 5.14: Training Loss for the 7000 items

5.3.2 BLEU Score Result Analysis on Test Set:

The result from other models like LLAMA and ChatGPT used to test and compare with NLTTA are obtained through prompt engineering. On conduction of BLEU score test on the 10% split of the dataset which was separated for test, NLTTA model got a score of 39.84 which is regarded as good translations. Conducting the same test with models used widely across the industries like Deepseek V3, ChatGPT, LLAMA 2, LLAMA 4 and Gemini 2.5 Flash we get the result displayed in the bar chat below:

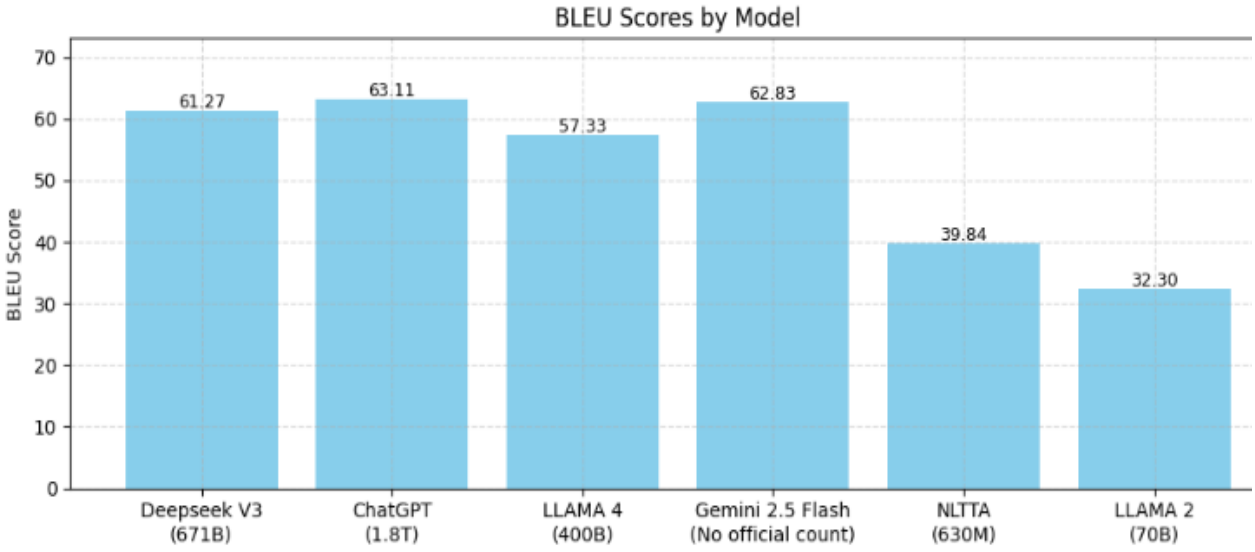

Figure 5.15: BLEU Score NLTTA vs Other Models on Test Set

While the chart clearly shows NLTTA achieving a score at 39.84, unlike the other models which are massive, general purpose LLMs with billions or even trillions of parameters (like Deepseek V3 at 671B or ChatGPT at 1.8T) and are trained on big diverse datasets for a longer period of time. The NLTTA model was developed as a specialized, highly focused solution (630M parameters). Achieving a robust BLEU score of nearly 40 points with a significantly smaller model and limited resources even outperforming LLAMA 2, which is a commendable technical achievement, demonstrating the project's success in building a functional, efficient translation solution tailored to specific needs, proving the viability of the approach within the project's defined scope.

5.3.3 BERT Score Result Analysis on Test Set:

On conduction of BERT score test on randomly selected and generated data, NLTTA model got a score of 0.9613 which is regarded as good translations. Conducting the same test with models used widely across the industries like Deepseek V3, ChatGPT, LLAMA 4, LLAMA 2 and Gemini 2.5 Flash we get the result displayed in the bar chat below:

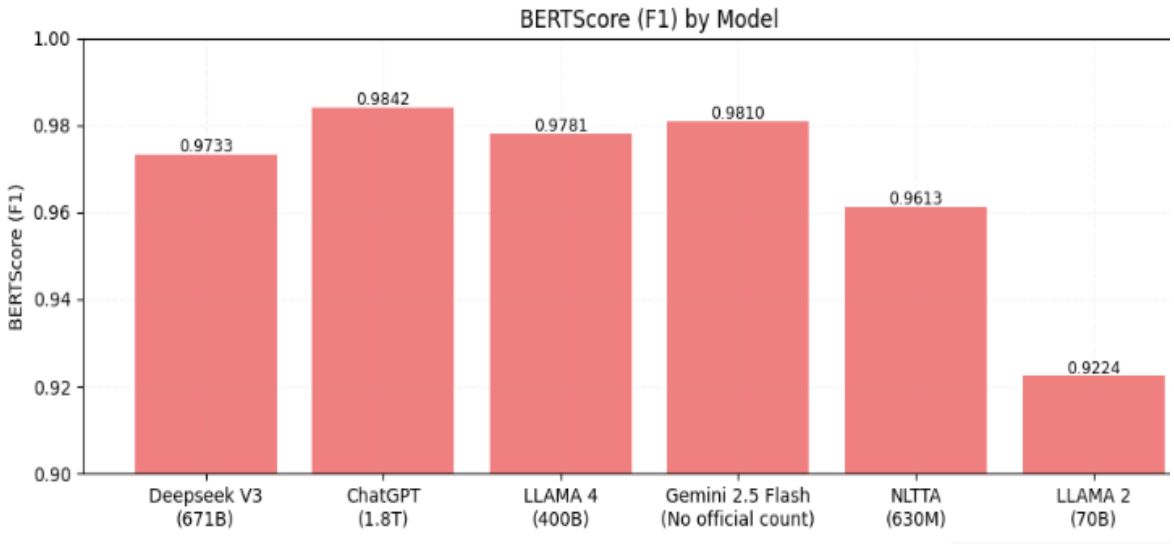

Figure 5.16: BERT Score NLTTA vs Other Models on Test Set

Among the models compared, this result must be evaluated with a realistic perspective on the model's structure and development focus. BERTScore, a metric based on contextual embeddings that measures semantic similarity, favors models trained on massive, diverse datasets, like the multi-billion and trillion-parameter models (e.g., ChatGPT at 1.8T) that dominate the top scores. Given that NLTTA is a highly specialized model with a significantly smaller scale (630M parameters) and was developed with limited resources focused on a niche translation task, NLTTA managed to outperform LLAMA 2 which is very considerably larger and expensive to run.

CHAPTER 6: CONCLUSION AND FUTURE RECOMMENDATIONS

6.1 Conclusion

The NLTTA web application successfully addresses the critical challenge of bridging the linguistic gap for Nepali users in the digital space. By leveraging advanced deep learning techniques, particularly transformer-based architectures like mBART, the system effectively processes Nepali inputs in both Devanagari and Romanized scripts, as well as mixed Nepali-English queries. NLTTA enables users to search for news content seamlessly, even when using informal, chat-style language. This significantly enhances digital inclusion, allowing individuals with limited English proficiency to access relevant online information easily. NLTTA ensures scalability for integrating additional features in the future.

Overall, the project can translate user queries and provide them with the most relevant news from the database. The project demonstrates promising results in improving digital accessibility and empowering a wider spectrum of Nepali speaking users to engage meaningfully with online platforms.

6.2 Future Recommendations

There are several potential future recommendations that could be implemented in the NLTTA project:

- **Expansion to Other Domains:** Integrate NLTTA with additional applications beyond news search, such as e-commerce platforms, government portals, and educational tools, to broaden its utility.
- **Enhanced Informal Language Handling:** Improve the model's capability to understand slang, dialect-specific expressions, and colloquial phrases commonly used in Nepali digital communication.
- **Multilingual and Multimodal Support:** Extend support to other regional languages spoken in Nepal and explore processing speech inputs for voice-based searches and commands.

- **Collaboration with Institutions:** Partner with educational and governmental institutions to promote NLTTA as a tool for enhancing digital literacy and bridging linguistic barriers nationwide.
- **Continuous Model Updating:** Regularly retrain and fine-tune the translation and retrieval models using updated datasets to maintain high accuracy and adapt to evolving language usage trends.
- **Mobile app:** Developing a mobile app could make the system more accessible and convenient for users who prefer to consume news on their mobile devices.

By implementing these future recommendations, the NLTTA project could continue to evolve and improve, providing a more valuable and engaging service to its users.

REFRENCES

[1] D. Bahdanau, K. Cho, and Y. Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate,” Sep. 2014, [Online]. Available: http://arxiv.org/abs/1409.0473

[2] S. Khanuja *et al.*, “MuRIL: Multilingual Representations for Indian Languages,” Mar. 2021, [Online]. Available: http://arxiv.org/abs/2103.10730

[3] Y. Liu *et al.*, “Multilingual Denoising Pre-training for Neural Machine Translation,” Jan. 2020, [Online]. Available: http://arxiv.org/abs/2001.08210

[4] T. Kudo and J. Richardson, “SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing,” Aug. 2018, [Online]. Available: http://arxiv.org/abs/1808.06226

[5] A. Vaswani *et al.*, “Attention Is All You Need,” *31st Conference on Neural Information Processing Systems*, Jun. 2017, [Online]. Available: http://arxiv.org/abs/1706.03762

[6] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space,” Jan. 2013, [Online]. Available: http://arxiv.org/abs/1301.3781

[7] S. Pudasaini, S. Shakya, A. Tamang, S. Adhikari, S. Thapa, and S. Lamichhane, “NepaliBERT: Pre-training of Masked Language Model in Nepali Corpus,” in *7th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud), I-SMAC 2023 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 325–330. doi: 10.1109/I-SMAC58438.2023.10290690.

APPENDIX

Figure 1: Home Page

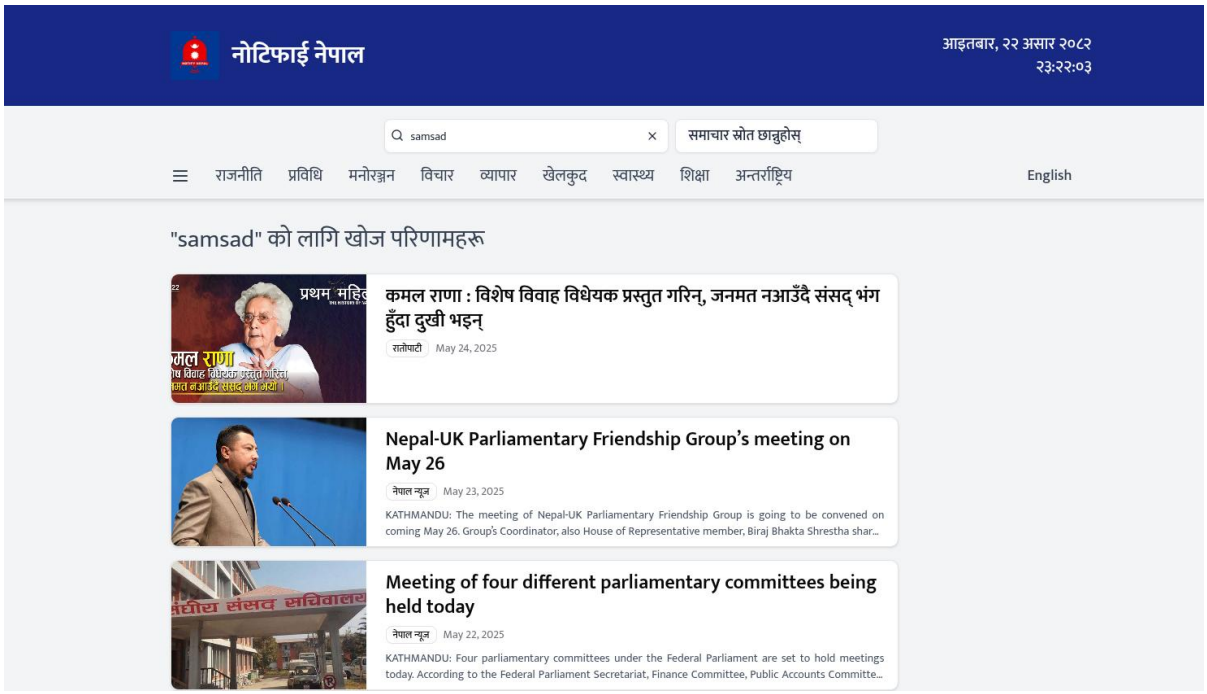

Figure 2:Searching samsad for parliament

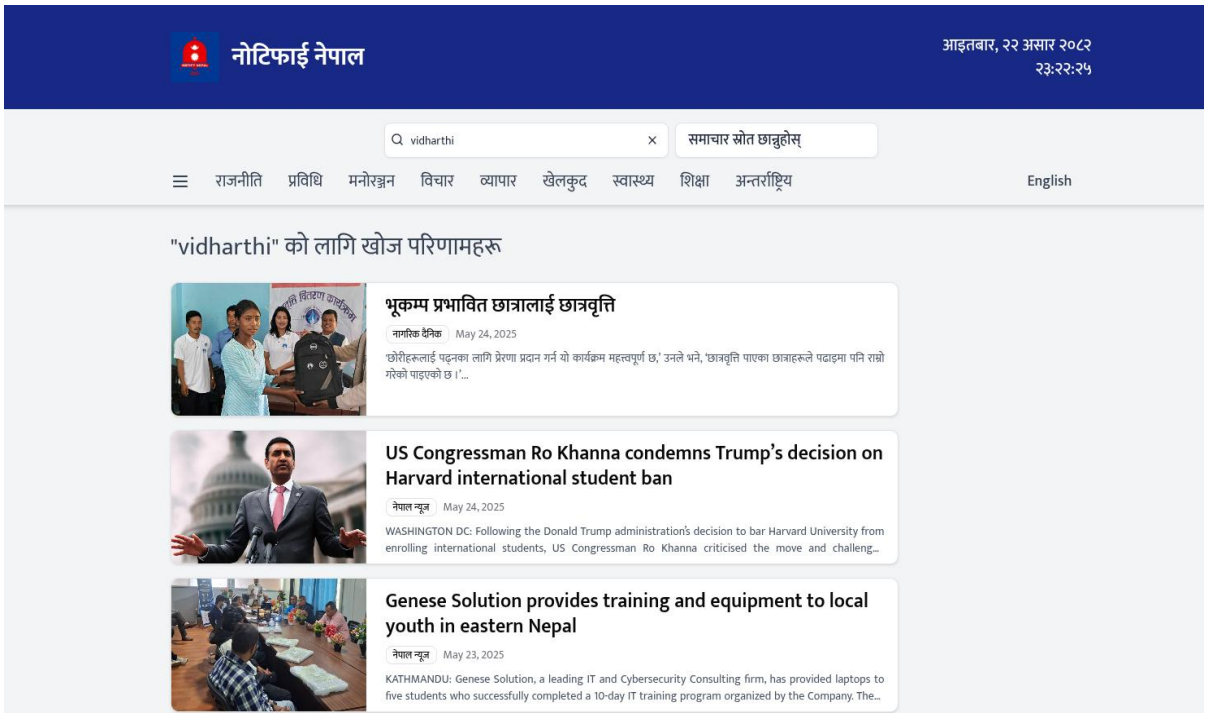

Figure 3:Searching Vidharthi for student

Figure 4:Article Page

Figure 5: Searching Ghar Mantri for HomeMinister

Figure 6: Why Nepali article comes with home minister

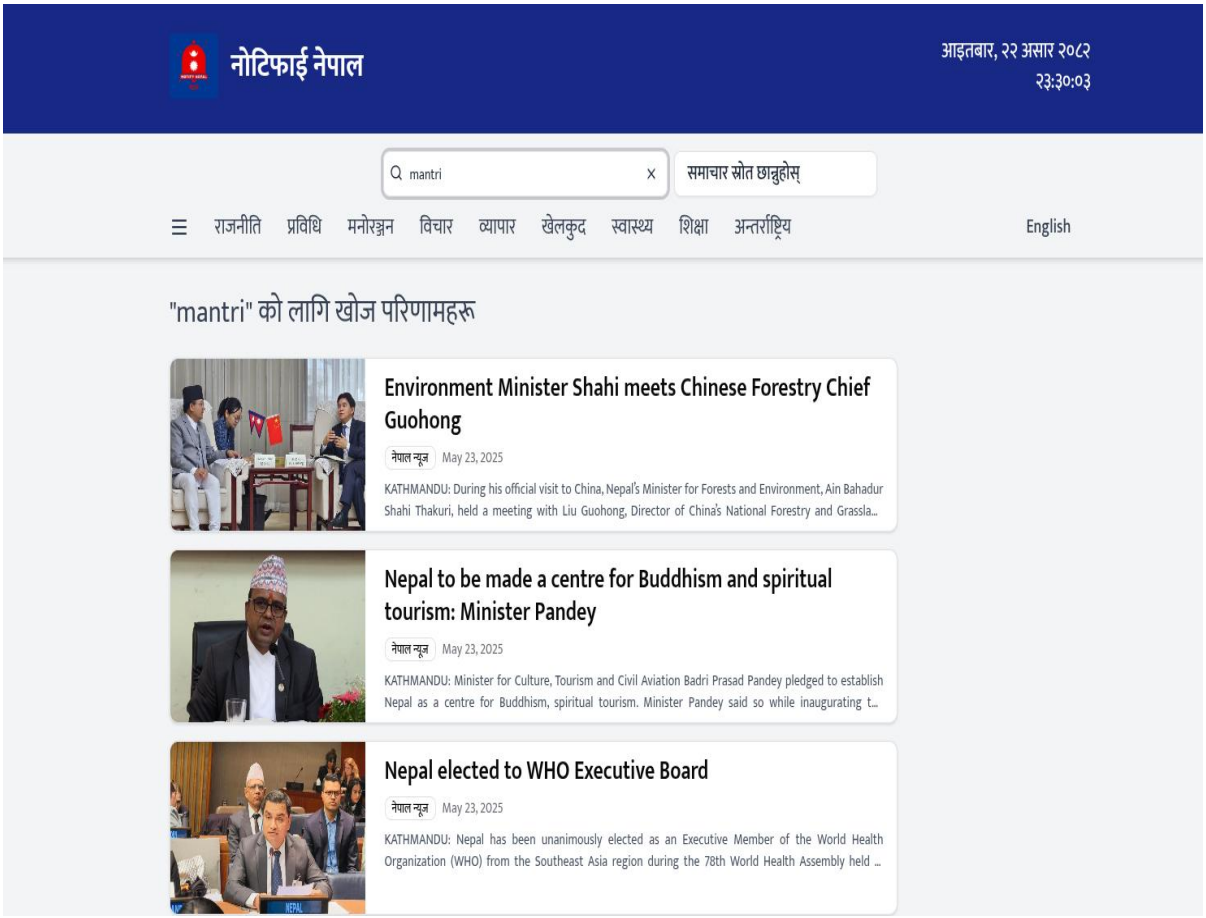

Figure 7: Searching 'mantri'

urls.py > ...

```python
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import FeedViewSet, FeedItemViewSet, TranslateViewSet

router = DefaultRouter()
router.register(r'feeds', FeedViewSet)
router.register(r'feed-items', FeedItemViewSet)
router.register('translate', TranslateViewSet, basename='translate')


urlpatterns = [
    path('', include(router.urls)),
]
```

Figure 8: Routing Structure of The Backend

```python
class Feed(models.Model):
    language = models.CharField(max_length=20)
    name = models.CharField(max_length=100)
    name_nepali = models.CharField(max_length=100)
    slug = models.SlugField(unique=True, max_length=255)
    website = models.URLField()
    logo_url = models.URLField()

    def __str__(self):
        return self.name


class FeedItem(models.Model):
    uuid = models.CharField(max_length=36, unique=True)
    slug = models.SlugField(unique=True, max_length=255)
    link = models.URLField()
    title = models.CharField(max_length=255)
    summary = models.TextField()
    content = models.TextField()
    image_url = models.URLField()

    is_translated = models.CharField(max_length=10, default='no')
    translated_content = models.TextField(null=True)

    published_at = models.DateTimeField()
    published_raw_nepali_date = models.CharField(max_length=50)
    fetched_at = models.DateTimeField()
    updated_at = models.DateTimeField(auto_now=True)

    feed = models.ForeignKey(Feed, on_delete=models.CASCADE, related_name='items')

    def __str__(self):
        return self.title
```

Figure 9: How the News RSS Feed are Structured and Stored

```python
serializers.py > ...
from rest_framework import serializers
from .models import Feed, FeedItem, Translate

class FeedSerializer(serializers.ModelSerializer):
    class Meta:
        model = Feed
        fields = '__all__'


class FeedItemSerializer(serializers.ModelSerializer):
    feed = FeedSerializer()

    class Meta:
        model = FeedItem
        fields = '__all__'

    def create(self, validated_data):
        feed_data = validated_data.pop('feed')
        feed, _ = Feed.objects.get_or_create(**feed_data)
        print(feed)
        return FeedItem.objects.create(feed=feed, **validated_data)


class TranslateSerializer(serializers.ModelSerializer):
    class Meta:
        model = Translate
        fields = '__all__'
        read_only_fields = ['created_at']
```

Figure 10: Serializers for News and Translated Text to Store in DB

```python
class TranslateViewSet(viewsets.ModelViewSet):
    queryset = Translate.objects.all().order_by('-created_at')
    serializer_class = TranslateSerializer

    def pre_process(self, text):
        text = text.lower()
        text = re.sub(r'[^\w\s]', '', text)
        text = re.sub(r'\s+', ' ', text).strip()

        return text

    def remove_word_prefixes(self, text, prefixes):
        pattern = r'^(' + '|'.join(re.escape(p) for p in prefixes) + r')\b\s*'
        return re.sub(pattern, '', text)

    def create(self, request, *args, **kwargs):
        try:
            source_text = request.data.get('source_text')
            if not source_text:
                return Response({"error": "source_text is required"},
                                status=status.HTTP_400_BAD_REQUEST)

            text = source_text
            inputs = tokenizer(self.pre_process(text), return_tensors="pt")
            generated = model.generate(
                **inputs,
                forced_bos_token_id=tokenizer.lang_code_to_id["en_XX"],
                max_length=128,
                num_beams=5
            )

            translated_text = self.remove_word_prefixes(
                str(tokenizer.decode(generated[0], skip_special_tokens=True)), ['e', 'ne'])

            serializer = self.get_serializer(data=request.data)
            serializer.is_valid(raise_exception=True)
            serializer.save(translated_text=translated_text)

            return Response(serializer.data, status=status.HTTP_201_CREATED)

        except Exception as e:
            return Response({
                "error": str(e),
                "trace": traceback.format_exc()
            }, status=500)
```

Figure 11: Translation API

LOG OF VISITS TO SUPERVISOR

Date	Remarks
2082/02/02	Abstract submission.
2082/03/20	Proposal submission for review and supervision.
2082/04/07	Discussion of project status and asked for suggestions.
2082/04/21	Showed working project demo.
2082/05/15	Initial report draft review and feedback session.
2082/06/27	Submitted final defense report for review and supervision.
2082/07/09	Report evaluations and further discussion.