Hindi Text Summarization Comparison Model using LLMs

Submitted in partial fulfillment of the requirements
for the degree of

B.E. Computer Engineering

By

Pranay Chavan	**48**	**212011**
Jennessa Rowena Dsouza	**61**	**212027**
Om Kamerkar	**63**	**212046**
Mansi Chaklashiya	**69**	**212010**

Guide
Mrs. Pradnya Sawant
Assistant Professor

Department of Computer Engineering
St. Francis Institute of Technology
(Engineering College)
An Autonomous Institute, Affiliated to University of Mumbai

2024-2025

SFIT LIBRARY 2025

CERTIFICATE

This is to certify that the project entitled **Hindi Text Summarization Comparison Model using LLMs** is a bonafide work of **Pranay Chavan 48, Jennessa Rowena Dsouza 61, Om Kamerkar 63 and Mansi Chaklashiya 69** submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of B.E. in Computer Engineering.

(Mrs. Pradnya Sawant)
Guide

(Dr. Kavita Sonawane)
Head Of Department

(Dr. Sincy George)
Principal

SFIT LIBRARY 2025

Project Approval Report for B.E.

This project report entitled ***Hindi Text Summarization Comparison Model using LLMs*** by ***Pranay Chavan 48, Jennessa Rowena Dsouza 61, Om Kamerkar 63 and Mansi Chaklashiya 69)*** is approved for the degree of ***B.E. in Computer Engineering***.

Examiners

1. _____ 30.4.25

2. _____ 30/4/25

Date: 30/04/2025

Place: Mumbai

SFIT LIBRARY 2025

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Pranay Chavan 48

Jennessa Rowena Dsouza 61

Om Kamerkar 63

Mansi Chaklashiya 69

Date: 30/04/2025

SFIT LIBRARY 2025

Abstract

Text summarization is the process of condensing long documents into concise summaries while retaining their essential information and context. While this task has seen significant progress in English, it remains relatively underexplored in Indian regional languages such as Hindi due to limited resources and tools. This research focuses on abstractive summarization of Hindi news articles using advanced transformer-based Large Language Models (LLMs). We evaluate and compare the performance of three models: IndicBART, mT5, and meta-llama/Llama-3.2-1B-Instruct, all fine-tuned on a curated Hindi news dataset.The models are assessed based on standard evaluation metrics including Precision, Recall, and F1-score, computed using BERT-based semantic similarity. Experimental results indicate that meta-llama/Llama-3.2-1B-Instruct significantly outperforms the other two models in generating fluent, coherent, and context-aware summaries. While IndicBART demonstrates better performance than mT5, it still lags behind the LLaMA-based model. Our findings underscore the effectiveness of fine-tuning LLMs for low-resource languages and emphasize the potential of domain-specific adaptation. Additionally, we explore the challenges faced in Hindi text summarization and propose directions for future work, such as the integration of reinforcement learning techniques or hybrid summarization methods.This study contributes to the growing body of research on Indian language processing and demonstrates the viability of LLMs in enhancing the quality of Hindi text summarization.

Keywords*: Hindi Text Summarization, Abstractive Summarization, Large Language Models, IndicBART, mT5, meta-llama/Llama-3.2-1B-Instruct, Transformer Models, Low-Resource Languages, Natural Language Processing, BERT Evaluation Metrics, Fine-tuning*

SFIT LIBRARY 2025

Contents

1 Introduction ***1***

1.1 Introduction 1

1.2 Background Study Terminologies/Definitions of New Terms 2

1.3 Fundamental Study Points of the Selected Topic and the Domain . 4

1.4 Identification of Challenges in the Selected Topic 6

1.5 Problem Statement and Proposed Solution 8

1.5.1 Problem Statement 8

1.5.2 Proposed Solution 9

1.6 Scope of the System 10

2 Review of Literature ***11***

2.1 Survey of Existing systems 11

2.2 Limitations of Existing systems or research gaps 14

2.3 Motivation 15

3 Proposed System: Analysis ***16***

3.1 Detailed explanation of Proposed system 16

3.1.1 Block diagram of Proposed system/ Workflow 16

3.1.2 Working Principle (Algorithm) 17

3.1.3 Phase/ Module - wise explanation (as per workflow) . . . 18

3.2 System Analysis 20

3.2.1 Functional Requirements 20

3.2.2 Non-Functional Requirements 20

3.2.3 Software and Hardware Requirements 22

SFIT LIBRARY 2025

3.2.4 Use Case Diagram 24
3.3 Proposed System: Analysis and Modeling 27
3.3.1 Activity Diagram 27
3.3.2 Sequence Diagram 28
3.3.3 DFD Diagram 29
3.3.4 Architectural View 33
3.3.5 Algorithms / Methodology 43
3.3.6 UI/UX DESIGN 45

4 Implementation Plan and Experimental Set up of the Proposed system 47
4.1 Experimental Set up 47
4.1.1 Details discussion of input/Dataset (along with screen-shot/ Samples) details about input to systems or selected data 47
4.1.2 Performance Evaluation Parameters 50
4.1.3 Special Requirement 50
4.2 Code 53
4.2.1 Model Implementation 53

5 Results and Discussions 59
5.1 Presentation and validation of the results for the proposed system 59
5.2 Comparative Analysis 59
*5.2.1 *Model Comparison: BERT Metrics* 60
5.2.2 Graphical Analysis of Indic BART Evaluation 61
5.2.3 Graphical Analysis of MT5 Evaluation 61
5.2.4 Graphical Analysis of LLAMA Evaluation 62
5.2.5 Comparative Analysis Of Evaluation Of All Three Models 62

6 Conclusion 63

Appendix-I 64

SFIT LIBRARY 2025

References ***65***

Acknowledgements ***66***

SFIT LIBRARY 2025

List of Figures

3.1 Block Diagram Of Hindi Text Summarization Model 16
3.2 Use Case Diagram for Hindi Text Summarization System (User) . 24
3.3 Use Case Diagram for System Administrator 25
3.4 Activity Diagarm Of Hindi Text Summarization using LLAMA Models 27
3.5 Sequence Diagram of Hindi Text Summarization Unsing LLMS Models 28
3.6 DFD Level 0: Overall System Overview 29
3.7 DFD Level 1: User Interaction 30
3.8 DFD Level 2: Detailed User Interaction 30
3.9 DFD Level 1: Admin Interaction 31
3.10 DFD Level 2: Admin Interaction 32
3.11 Architecture of Indic BART 33
3.12 Architecture of mT5 37
3.13 Architecture of LLaMA 40
3.14 Hindi Text Summarizer - Input Text and Summary Settings Screen . 46
3.15 Generated Summaries Display Screen (IndicBART, mT5, Llama 3.2 1B) 46

4.1 Dataset Overview 47
4.2 Dataset Overview 48

5.1 Results of all Evaluation Metrics of all Models 59
5.2 Indic BART Evaluation Results 61
5.3 MT5 Evaluation Results 61

SFIT LIBRARY 2025

5.4 LLAMA 3.2.1 Instruct B Evaluation Results 62
5.5 Comparative Analysis of All Three Models(Indic BART, MT5 and LLAMA 62

SFIT LIBRARY 2025

List of Tables

2.1 Summary of Literature Review 13

3.2 Key Formulas and Equations for IndicBART 36

3.4 Formulas Used in LLaMA Architecture 42

5.1 Comparison of BERT Precision, Recall, and F1 Score Across Models 60

SFIT LIBRARY 2025

Chapter 1

Introduction

1.1 Introduction

Hindi text summarization is an essential task in Natural Language Processing (NLP), focusing on the automatic generation of concise, coherent, and meaningful summaries from lengthy Hindi documents, such as news articles. Despite advancements in summarization for English, the progress in Indian regional languages, especially Hindi, remains limited due to the lack of annotated datasets and model resources.This project investigates abstractive summarization, a technique that involves generating novel sentences rather than simply extracting them from the original text. We explore the effectiveness of three transformer-based Large Language Models (LLMs): IndicBART, mT5, and meta-llama/Llama-3.2-1B-Instruct. Each model was fine-tuned on a curated Hindi news dataset to improve their ability to produce fluent and contextually accurate summaries.The models were evaluated using semantic similarity-based metrics—Precision, Recall, and F1-score—derived from BERT embeddings. Experimental results show that meta-llama/Llama-3.2-1B-Instruct delivers superior performance, demonstrating a stronger grasp of contextual understanding and linguistic nuances in Hindi. IndicBART outperforms mT5, reaffirming its relevance and capability in handling Indian languages.This study highlights the importance of domain-specific fine-tuning when applying LLMs to low-resource languages. Furthermore, we analyze each model's limitations and propose avenues for improvement, including reinforcement learning strategies

SET LIBRARY 2025

and hybrid summarization approaches. Ultimately, this work contributes to the growing field of Hindi NLP and lays the groundwork for developing more efficient and accurate summarization tools for Indian regional languages.

1.2 Background Study Terminologies/Definitions of New Terms

I. **Text Summarization**

a. **Definition**: The process of creating a concise version of a text that retains the original's key points and essential information.

b. **Types**: There are two primary approaches:

- **Extractive Summarization**: Selecting sentences directly from the text.
- **Abstractive Summarization**: Generating summaries that may not contain exact phrases from the original text.

II. **Transformer Model**

a. **Definition**: A deep learning model architecture introduced by Vaswani et al., which uses self-attention mechanisms to process sequences of data, especially effective in NLP tasks.

b. **Relevance**: Transformer models form the backbone of advanced text summarization tasks, enabling context-aware understanding of long texts.

III. **Indic BART**

a. **Definition**: A variant of the BART (Bidirectional and Auto-Regressive Transformer) model, fine-tuned specifically for Indian languages.

b. **Role in the Project**: Indic BART is used in this project to generate high-quality summaries in Hindi, as it has been optimized for the linguistic patterns of Indian regional languages.

IV. **mT5 (Multilingual Text-to-Text Transfer Transformer)**

a. **Definition**: A multilingual model capable of performing various NLP tasks by treating each task as a text-to-text transformation, allowing it to handle multiple languages effectively.

SETT LIBRARY 2025

b. **Significance in Summarization**: mT5's multilingual capabilities make it suitable for summarizing Hindi texts, supporting the project's goal of regional language NLP.

V. **LLaMA (Large Language Model Meta AI)**

a. **Definition**: LLaMA is a family of large-scale foundation models developed by Meta, designed for various NLP tasks. It is optimized for efficiency and open research, making it a strong alternative to proprietary models.

b. **Significance in Summarization**: LLaMA's capability to process and generate human-like text enables effective Hindi text summarization. Its open-source nature allows for fine-tuning, making it adaptable to the project's specific needs in regional language NLP.

VI. **BERTScore Metric**

a. **Definition**: An evaluation metric for text generation tasks like summarization that measures semantic similarity between generated text and reference text using BERT embeddings.

b. **Purpose in Evaluation**: In this project, BERTScore is used to assess the quality of summaries generated by different models, evaluating how closely they align with the reference summaries in terms of meaning and content.

VII. **Precision, Recall, and F1-Score**

a. **Precision**: The proportion of relevant items among retrieved items.

b. **Recall**: The proportion of relevant items that have been retrieved over the total amount of relevant items.

c. **F1-Score**: The harmonic mean of Precision and Recall, used to assess the overall accuracy of a model.

d. **Usage**: These metrics are employed in this project to quantitatively evaluate the performance of the summarization models.

SFIT LIBRARY 2025

VIII. **Preprocessing Techniques**

a. **Definition**: The set of steps taken to prepare raw text data for model input, often including tokenization, removal of stop words, and normalization.

b. **Hindi-Specific Preprocessing**: Techniques that are adjusted to handle Hindi's syntactic and morphological nuances, such as handling compound words, gender-specific language forms, and script peculiarities.

1.3 Fundamental Study Points of the Selected Topic and the Domain

I. **Natural Language Processing (NLP)**

a. **Definition**: A field of artificial intelligence focused on enabling computers to understand, interpret, and generate human language.

b. **Relevance to Project**: NLP provides the techniques and tools necessary for building models that can analyze and generate language-based outputs, forming the backbone of this Hindi text summarization project.

II. **Challenges in Hindi NLP**

a. **Complexity of Syntax and Morphology**: Hindi, like many regional languages, presents unique grammatical structures and word inflections that require specialized processing.

b. **Resource Scarcity**: Unlike English, Hindi NLP suffers from a lack of large-scale annotated datasets, models, and resources tailored to its linguistic features.

III. **Text Summarization**

a. **Definition**: The process of creating a concise version of a text while retaining its key points and information.

b. **Types**:

- **Extractive Summarization**: Involves selecting and piecing together sentences from the original text.

SFIT LIBRARY 2025

- **Abstractive Summarization**: Involves generating new sentences that encapsulate the meaning of the original text.

c. **Application in Hindi**: Given Hindi's linguistic complexity, abstractive summarization poses challenges in terms of grammatical coherence and maintaining semantic integrity.

IV. **Transformer Architecture**

a. **Self-Attention Mechanism**: The foundation of the transformer model, allowing it to weigh the importance of different words in a sentence, thus capturing context more effectively than previous models.

b. **Role in NLP**: Transformers represent a breakthrough in NLP tasks, including summarization, due to their ability to handle long-range dependencies within text.

V. **Preprocessing Techniques for Hindi Text**

a. **Tokenization**: Separates text into individual words or subwords, crucial for feeding text into models. In Hindi, this often includes handling compound words and transliterations.

b. **Normalization and Stop Word Removal**: Removes commonly used words that may not carry significant meaning, but in Hindi, extra care is required due to variations in script and form.

VI. **Evaluation Metrics for Summarization**

a. **BERTScore Metric**: Measures semantic similarity between generated summaries and reference summaries, assessing how accurately the model captures the essence of the original text.

b. **Precision, Recall, and F1-Score**: Standard metrics for evaluating summarization models, used to assess the quality and relevance of generated summaries.

SFIT LIBRARY 2025

VII. **Multilingual Models and Transfer Learning**

a. **Multilingual Models**: Models like mT5 are designed to work across multiple languages, leveraging cross-linguistic data to improve performance in lower-resource languages.

b. **Transfer Learning**: Fine-tuning pre-trained models on Hindi-specific datasets enables these models to adapt to the linguistic nuances of Hindi, a critical step for achieving accurate summarization.

VIII. **Application of Summarization in Regional Languages**

a. **Potential Use Cases**: Summarization can be applied in fields like education, media, government, and customer service, where summarizing long Hindi documents can improve accessibility and efficiency.

b. **Impact on NLP in India**: Advancements in Hindi summarization could set a precedent for NLP applications in other regional languages, promoting inclusivity in digital information access.

1.4 Identification of Challenges in the Selected Topic

I. **Lack of Large-Scale Hindi Datasets**

- High-quality, annotated datasets for Hindi summarization are limited compared to English.
- The scarcity of data affects the fine-tuning of transformer models like Indic BART and mT5.
- Collecting and preprocessing Hindi text data is time-consuming and requires linguistic expertise.

II. **Complexity of Hindi Grammar and Morphology**

- Hindi has rich inflections, compound words, and gender-specific verbs.
- Handling morphological variations while maintaining summary coherence is challenging.

SFIT LIBRARY 2025

- Preprocessing must be adapted to preserve meaning while simplifying text for summarization.

III. **Translation and Code-Switching Issues**

- Many Hindi texts contain mixed-language content, especially Hindi-English code-switching.
- Summarization models struggle with maintaining linguistic consistency in such cases.
- Special preprocessing techniques are needed to handle transliterations and mixed-script text.

IV. **Evaluation Metric Limitations**

- ROUGE and BERTScore may not always reflect the true quality of Hindi summaries.
- Word-order variations and synonym usage in Hindi can result in lower metric scores.
- Human evaluation remains necessary, increasing subjectivity and effort in performance assessment.

V. **Loss of Context in Abstractive Summarization**

- Transformer models sometimes generate summaries that miss critical information.
- Unlike extractive summarization, abstractive models may introduce factual inconsistencies.
- Ensuring factual correctness while generating concise summaries remains an open challenge.

VI. **Computational Resource Constraints**

- Fine-tuning large models like LLaMA and mT5 requires high-end GPUs and extensive memory.

SFIT LIBRARY 2025

- Training on Hindi-specific corpora can be expensive and time-consuming.
- Limited access to computational resources may restrict experimentation with larger architectures.

VII. **Handling Named Entities and Domain-Specific Terms**

- Hindi text often includes region-specific proper nouns and technical terms.
- Named Entity Recognition (NER) models trained on English do not perform well for Hindi.
- Incorrect entity recognition can distort the meaning of the summary.

1.5 Problem Statement and Proposed Solution

1.5.1 Problem Statement

The growing demand for summarization tools in Indian regional languages, especially Hindi, remains largely unmet due to the scarcity of annotated datasets, complex linguistic structure, and lack of tailored models. Hindi, being morphologically rich and syntactically different from English, poses challenges in generating accurate and context-aware summaries. Existing solutions are either extractive or English-centric, failing to produce semantically rich summaries for Hindi texts.This study focuses on developing a Hindi text summarization system that leverages the power of large language models (LLMs). Specifically, the models explored are IndicBART (Indic Bidirectional and Auto-Regressive Transformer), mT5 (Multilingual Text-to-Text Transfer Transformer), and LLaMA 3.2B Instruct model. These LLMs will be fine-tuned on curated Hindi datasets to generate abstractive summaries that preserve meaning and grammatical structure.The objective is to create a robust summarization system that bridges the gap between English and Hindi NLP tools by evaluating and comparing these models using metrics such as Precision, Recall, and F1-score. The resulting system can support real-world applications in news summarization, document abstraction, academic writing, and public administration.

SFIT LIBRARY 2025

1.5.2 Proposed Solution

To address the challenges of Hindi text summarization, we propose a solution that utilizes three pre-trained Large Language Models (LLMs): IndicBART, mT5, and LLaMA 3.2B Instruct. These models, already trained on large multilingual corpora, are further fine-tuned on a domain-specific Hindi dataset prepared by us to enhance their abstractive summarization capabilities.

The proposed solution follows these key steps:

a. **Dataset Preparation:** Collect and preprocess a suitable Hindi text dataset, including cleaning, tokenization, and formatting it for fine-tuning.

b. **Model Acquisition:** Download the pre-trained IndicBART, mT5, and LLaMA 3.2B Instruct models from their respective sources.

c. **Fine-Tuning:** Fine-tune each of the three models on the prepared Hindi dataset to optimize them for the summarization task.

d. **Training and Optimization:** Train the models further by adjusting hyperparameters to achieve the best performance.

e. **Performance Analysis:** Evaluate the fine-tuned models using Precision, Recall, F1-score, and BERTScore metrics.

f. **Comparative Study:** Perform a detailed comparison of the models based on their evaluation metrics to determine their relative performance.

g. **Visualization of Results:** Visualize the performance outcomes using appropriate graphs and charts for better interpretation.

Through this step-by-step approach, we aim to develop a robust Hindi text summarization system that bridges the gap between English and Hindi summarization tools.

SFIT LIBRARY 2025

1.6 Scope of the System

The proposed Hindi text summarization system has a wide range of applications and potential benefits. Below are the key aspects that define its scope:

a. **Enhanced Accessibility for Hindi Content:** The system enables efficient summarization of Hindi text, making it accessible for users who need concise versions of lengthy documents, including students, researchers, and professionals.

b. **Application in News and Media:** Journalists and media professionals can use this tool to generate quick summaries of Hindi news articles, helping readers grasp key points without reading entire reports.

c. **Legal and Academic Document Summarization:** The system can assist in summarizing lengthy legal and academic documents, allowing lawyers and scholars to extract essential information efficiently.

d. **Support for Government and Administrative Reports:** Government agencies dealing with vast amounts of Hindi text can use this tool to generate brief yet informative summaries for policy-making, public communication, and archival purposes.

e. **Integration with AI-Powered Assistants:** The summarization model can be integrated into AI-driven virtual assistants and chatbots to provide users with quick, contextually relevant responses in Hindi.

f. **Future Expansion for Multilingual Capabilities:** Although currently focused on Hindi, the system can be extended to support other Indian regional languages, broadening its impact on Natural Language Processing (NLP) applications in India.

SFIT LIBRARY 2025

Chapter 2

Review of Literature

2.1 Survey of Existing systems

Text summarization is gaining importance in NLP, but low-resource languages like Hindi are less explored. This study uses IndicBART, a lightweight version of BART, optimized for Indian languages. IndicBART follows a sequence-to-sequence Transformer design and is fine-tuned for Hindi summarization. It is more efficient than models like mBART and mT5. The model was trained on 7,600 samples and validated on 357 after preprocessing. Training was done using PyTorch and Hugging Face on a 12GB Nvidia K80 GPU. Resource limitations were managed with smaller batch sizes, gradient accumulation, and warmup steps. Evaluation used ROUGE, achieving a ROUGE-1 F-score of 0.544. The results show IndicBART's ability to generate accurate and concise Hindi summaries. This highlights its promise for low-resource language applications.**[1]**

Multilingual models like mBART, mT5, and IndicBART enable summarization in low-resource Indian languages. However, limited dataset availability hampers training effectiveness. This study by Taunk and Varma evaluates IndicBART and mT5 on datasets in Hindi, Gujarati, and English. Indian news articles often involve code-switching and script-mixing, posing unique summarization challenges. The dataset includes articles, headlines, and summaries, with a custom validation set created due to missing reference summaries. Both models were fine-tuned using Hugging Face and PyTorch, with six encoder and

SFIT LIBRARY 2025

decoder layers. Data augmentation (3x and 5x) significantly improved performance. ROUGE scores were used for evaluation. Results showed that fine-tuning and augmentation enhance multilingual summarization. The study highlights the effectiveness of IndicBART and mT5 for Indian languages..**[2]** Krishnakumar et al. investigate the use of pre-trained models for summarizing Indian languages, addressing the challenges of code-mixed and script-mixed articles in Hindi, Gujarati, and English. This study, conducted as part of the ILSUM task in FIRE 2022, evaluates models such as mT5_m2m_CrossSum, XL-Sum, and BERT-based architectures. Among these, mT5_m2m_CrossSum demonstrated the best results, especially for Hindi and Gujarati, achieving strong ROUGE and LaSE scores. XL-Sum also provided concise and high-quality summaries, excelling in English summarization. BERT, known for its bidirectional encoding, achieved high performance across multiple NLP tasks and handled code-mixed summaries effectively. Overall, the findings suggest that cross-lingual, pre-trained models significantly enhance summarization accuracy for low-resource Indian languages, overcoming challenges posed by language mixing.

SFIT LIBRARY 2025

Paper	Review of Literature	Techniques/Models Used	Methodology
[1]	Focuses on IndicBART for Hindi abstractive summarization, targeting low-resource languages with tailored implementation.	• Preprocessing: **HTML** tag removal, cleaning • Model: **IndicBART** • Training: Fine-tuning • **PyTorch**, Hugging Face scripts	IndicBART fine-tuned on Hindi datasets using ROUGE metrics via PyTorch and HuggingFace.
[2]	Comparative evaluation of IndicBART and mT5 on Indian languages using data augmentation.	• Abstractive Summarization • Data Augmentation • Models: **IndicBART**, **mT5** • Tools: Hugging Face API, **PyTorch**	Evaluated IndicBART and mT5 with data augmentation; used ROUGE metrics for multilingual performance comparison.
[3]	Benchmarks various models (mT5, CrossSum, BERT, etc.) across English, Hindi, and Gujarati; strong cross-lingual focus.	• Abstractive Summarization • Evaluation Metrics: ROUGE, LaSE • Models: mT5-m2m_CrossSum, XL-Sum, **BERT**, mT5-small	Assessed multilingual models across diverse datasets using ROUGE and LaSE for performance evaluation.

Table 2.1: Summary of Literature Review

SHIT LIBRARY 2025

2.2 Limitations of Existing systems or research gaps

1. **Limited Datasets for Indian Languages:** Existing work highlights a scarcity of large, diverse, and high-quality datasets in Indian languages. This data limitation affects both the training and testing phases of summarization models, particularly for languages like Hindi, Gujarati, and other low-resource Indian languages.

2. **Code-Mixing and Script-Mixing Challenges:** Indian languages often incorporate code-mixing (e.g., Hindi-English or Gujarati-English) and script-mixing, which complicates summarization tasks. Models typically trained on monolingual or high-resource languages struggle to capture these nuances, leading to inaccuracies in summaries.

3. **Lack of Language-Specific Fine-Tuning:** While multilingual models like mT5, IndicBART, and XL-Sum perform well in general, they are not fully optimized for Indian languages. The models often require more fine-tuning or data augmentation tailored specifically to the linguistic structures and idiomatic expressions in Indian languages.

4. **Low Focus on Abstractive Summarization:** Much of the existing work still focuses on extractive summarization, which often lacks the flexibility of abstractive models. However, Indian language summaries generated through extractive methods may be limited in coherence, contextual relevance, or brevity.

5. **Evaluation Metric Limitations:** ROUGE and F1 scores, commonly used for model evaluation, may not accurately reflect the quality of generated summaries for Indian languages. There is a need for more nuanced and language-sensitive evaluation metrics like LaSE, particularly for code-mixed text.

SFIT LIBRARY 2025

6. **Dependence on Large, Resource-Heavy Models:** State-of-the-art models like mT5 and IndicBART require extensive computational resources, making them challenging to deploy in real-time applications or in resource-limited settings typical for many Indian institutions.

2.3 Motivation

1. **Growing Demand for Regional NLP Tools:** As digital content in Hindi expands, there is an increasing need for efficient summarization tools to process and simplify information for Hindi-speaking audiences.

2. **Limitations in Existing Resources:** While English text summarization models are well-developed, Hindi lacks comparable tools, which limits accessibility and usability for native speakers.

3. **Handling Linguistic Complexity:** Hindi's unique grammatical structure and morphology present challenges for standard models, motivating the development of specialized approaches to improve summarization accuracy.

4. **Enhanced Accuracy and Relevance:** By implementing models like IndicBART and mT5 with language-specific fine-tuning, we aim to boost accuracy and coherence in Hindi summaries.

5. **Broader NLP Advancements for Low-Resource Languages:** This project contributes to the broader field of NLP by advancing technology for low-resource languages, making Hindi text summarization more effective and accessible.

SFIT LIBRARY 2025

Chapter 3

Proposed System: Analysis

3.1 Detailed explanation of Proposed system

3.1.1 Block diagram of Proposed system/ Workflow

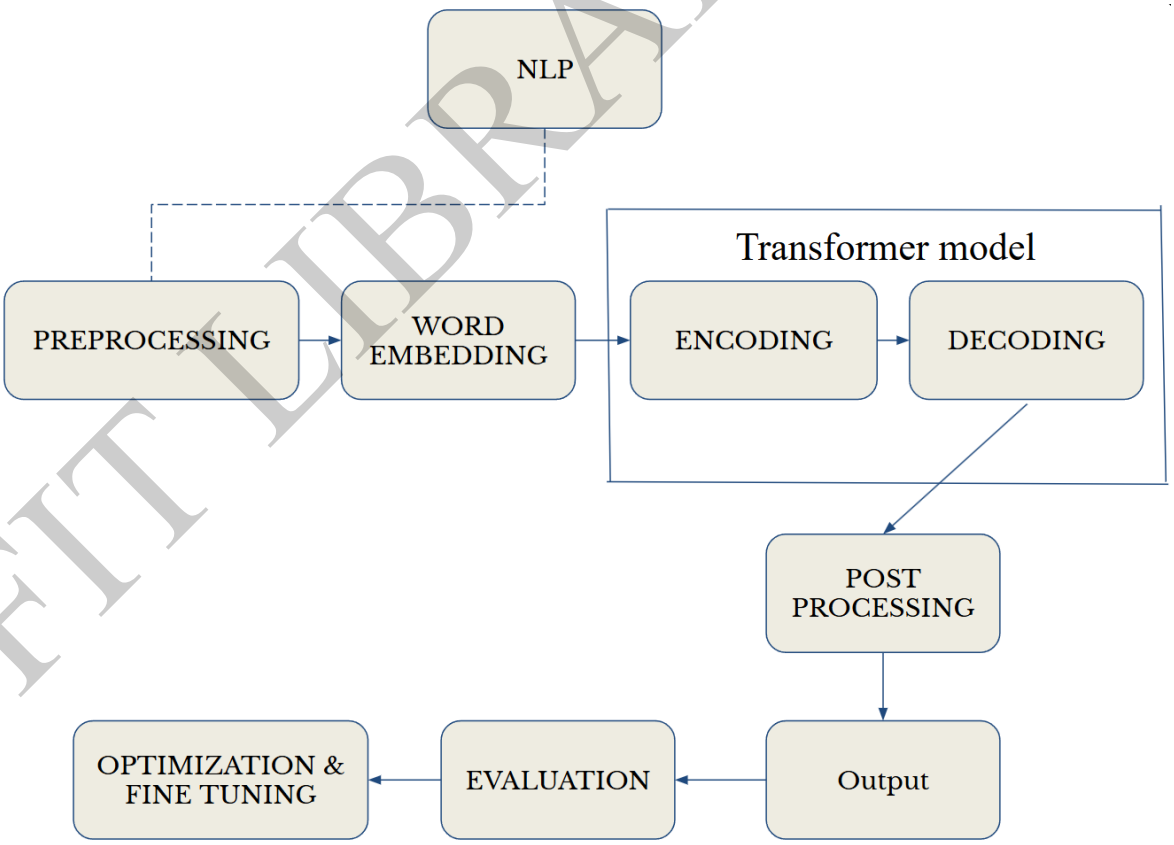

Figure 3.1: Block Diagram Of Hindi Text Summarization Model

SFIT LIBRARY 2025

3.1.2 Working Principle (Algorithm)

The Hindi Text Summarization system uses a structured pipeline incorporating LLaMA 3.3, IndicBART, and mT5. The complete workflow involves environment setup, data preprocessing, model fine-tuning, and evaluation.

1. **Environment Setup and Dependency Installation**
   - Create a virtual environment and install necessary libraries using pip.
   - Ensure GPU support with CUDA/cuDNN and load models using Hugging Face transformers.
2. **Data Acquisition and Preprocessing**
   - Download the ILSUM-23 dataset via gdown.
   - Clean text using emoji and indic-num2words, remove noise, lowercase, and tokenize.
   - Split dataset into training, validation, and testing sets (80-10-10).
3. **Model Selection and Preparation**
   - Use IndicBART, mT5, and LLaMA 3.3 pre-trained models.
   - Prepare dataset with tokenization, padding, truncation, and attention masks.
4. **Model Fine-Tuning**
   - Define training parameters: epochs, batch size, learning rate.
   - Train using the Trainer API with backpropagation and learning rate scheduling.
5. **Inference and Summary Generation**
   - Tokenize input, generate summaries, decode token IDs to text.
   - Use beam search and length/repetition constraints for quality.

SFIT LIBRARY 2025

6. **Evaluation**
    - Use BERT-based metrics: Precision, Recall, and F1-score to compare models.
7. **Deployment and Future Enhancements**
    - Deploy the best model; consider domain-specific tuning and expanding to other Indian languages.

3.1.3 Phase/ Module - wise explanation (as per workflow)

The proposed system involves several phases for efficient Hindi text summarization. Below is a brief overview of each phase:

1. **Preprocessing:**
    - **Description:** Includes tokenization, removing stop words, stemming, and lemmatization.
    - **Relevance:** For Hindi, it includes handling compound words, diacritics, and script-specific features.
2. **Word Embedding:**
    - **Description:** Converts text into numerical representations using embeddings like Word2Vec, GloVe, or BERT-based embeddings.
    - **Relevance:** Embeddings must capture semantic nuances of Hindi, particularly through pre-trained or multilingual embeddings.
3. **Transformer Model:**
    (a) **Encoding:**
        - **Description:** Uses the encoder part of Transformer to generate context-aware representations.
        - **Relevance:** Encoder must handle Hindi's syntactic and semantic structure.

SFIT LIBRARY 2025

(b) **Decoding:**

- **Description:** Generates summaries from the encoded data using the Transformer decoder.
- **Relevance:** Ensures the output text is coherent and contextually accurate in Hindi.

4. **Post Processing:**

- **Description:** Refines the output with detokenization, grammar correction, and readability improvements.
- **Relevance:** Ensures the summarized text is fluent and properly formatted for Hindi.

5. **Evaluation:**

- **Description:** Evaluates summary quality using metrics like ROUGE, BLEU, or human evaluation.
- **Relevance:** Requires metrics tailored for Hindi, with possible human evaluation for contextual appropriateness.

6. **Optimization & Fine Tuning:**

- **Description:** Optimizes performance via hyperparameter tuning, transfer learning, and domain-specific fine-tuning.
- **Relevance:** Fine-tuning on Hindi datasets enhances model performance.

7. **Output:**

- **Description:** Generates the final summary or comparative results.
- **Relevance:** The output should be a concise, accurate summary for practical use.

SFIT LIBRARY 2025

3.2 System Analysis

3.2.1 Functional Requirements

1. **Data Preprocessing:** The system should be capable of reading and processing raw Hindi text from datasets (train and test) by removing unnecessary elements (such as stop words, special characters, and punctuation) and converting the text into a suitable format for model input.

2. **Text Summarization:** The core functionality of the system is to generate abstractive summaries for the given Hindi articles. The system should leverage language models like IndicBART, mT5, and custom transformers to generate meaningful and contextually relevant summaries.

3. **Model Training:** The system should support the training of models using the provided training dataset (train.csv) to fine-tune pre-trained models. This includes configuring hyperparameters, training the models, and saving them for future use.

4. **Evaluation:** The system should evaluate the performance of the summarization models using metrics such as Precision, Recall, F1-score, and others that ensure the generated summaries are accurate and reliable. The evaluation should be done on the test dataset (test.csv).

5. **User Interface:** The system should have a user-friendly interface that allows users to input new Hindi articles, receive summaries, and view performance metrics. It should be simple to navigate and intuitive for non-technical users.

3.2.2 Non-Functional Requirements

1. **Performance:** The system should be able to generate summaries quickly, with minimal latency, especially when deployed in real-time applications.

2. **Scalability:** The system should be scalable, meaning that it can handle larger datasets in the future. As the dataset grows, the system should be

SFIT LIBRARY 2025

able to process and generate summaries without significant performance degradation.

3. **Accuracy:** The system should strive for high accuracy in text summarization, producing coherent and contextually relevant summaries that capture the key points of the original text.

SFIT LIBRARY 2025

3.2.3 Software and Hardware Requirements

1. **Software Requirements**

   (a) **Operating System:**

   - Windows can be used, but Linux is generally preferred for development.
   - Windows Subsystem for Linux (WSL) is a good alternative.

   (b) **Programming Language:**

   - Python (3.8 or higher) is required for working with deep learning libraries.

   (c) **Python Libraries:**

   - PyTorch (1.10 or higher): Deep learning framework.
   - Transformers (Hugging Face): For pre-trained models and tokenizers.
   - Accelerate: For simplified distributed training.
   - Pandas: For data manipulation.
   - Scikit-learn: For data splitting and other utilities.
   - Bert-score: For BERT score evaluation.
   - Emoji: For emoji handling.
   - Indic-num2words: For Hindi number-to-word conversion.
   - gdown: For downloading data from Google Drive.
   - Matplotlib: For plotting the distribution of data.

2. **Hardware Requirements**

   (a) **CPU:**

   - A multi-core CPU (at least 4 cores) is recommended.

   (b) **GPU (Recommended for Faster Training):**

   - NVIDIA GPU with at least 8GB VRAM (16GB or more is highly recommended for larger models and datasets).

SFIT LIBRARY 2025

- A powerful GPU significantly accelerates the training process.
- NVIDIA Tesla T4, NVIDIA A100, or similar GPUs are ideal for large-scale training.

(c) **RAM:**

- 16GB or more RAM is recommended.
- 32GB or more is preferred for larger models and datasets.

(d) **Storage:**

- SSD (Solid State Drive) is recommended for faster data loading and processing.
- Sufficient storage space is needed for the dataset, model checkpoints, and other files.

(e) **Internet Connection:**

- A stable and reasonably fast internet connection is required for downloading datasets.

SFIT LIBRARY 2025

3.2.4 Use Case Diagram

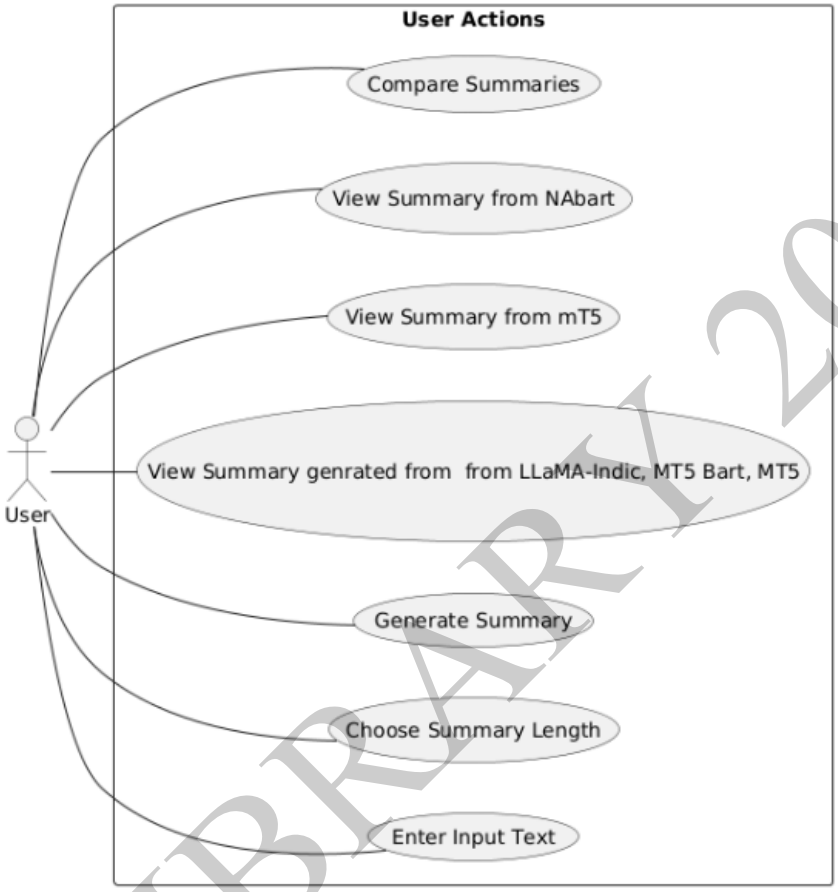

Figure 3.2: Use Case Diagram for Hindi Text Summarization System (User)

The use case diagram represents user interactions with the Hindi text summarization system using models like Indic BART, mT5, and LLaMA. It includes input, selection, summary generation, and comparison functionalities.

- **Enter Input Text:** User provides Hindi text that needs to be summarized.
- **Choose Summary Length:** User selects the desired length of the summary (short, medium, long).
- **Generate Summary:** User initiates the summarization process.
- **View Summary generated from LLaMA-Indic, MT5 Bart, MT5:** User views summaries produced by LLaMA-Indic, mT5-BART, and mT5 models.

SFIT LIBRARY 2025

- **View Summary from mT5:** User separately views the summary generated only by the mT5 model.
- **View Summary from NAbart:** User views the summary generated by the NAbart model.
- **Compare Summaries:** User compares the outputs from different models to evaluate performance and quality.

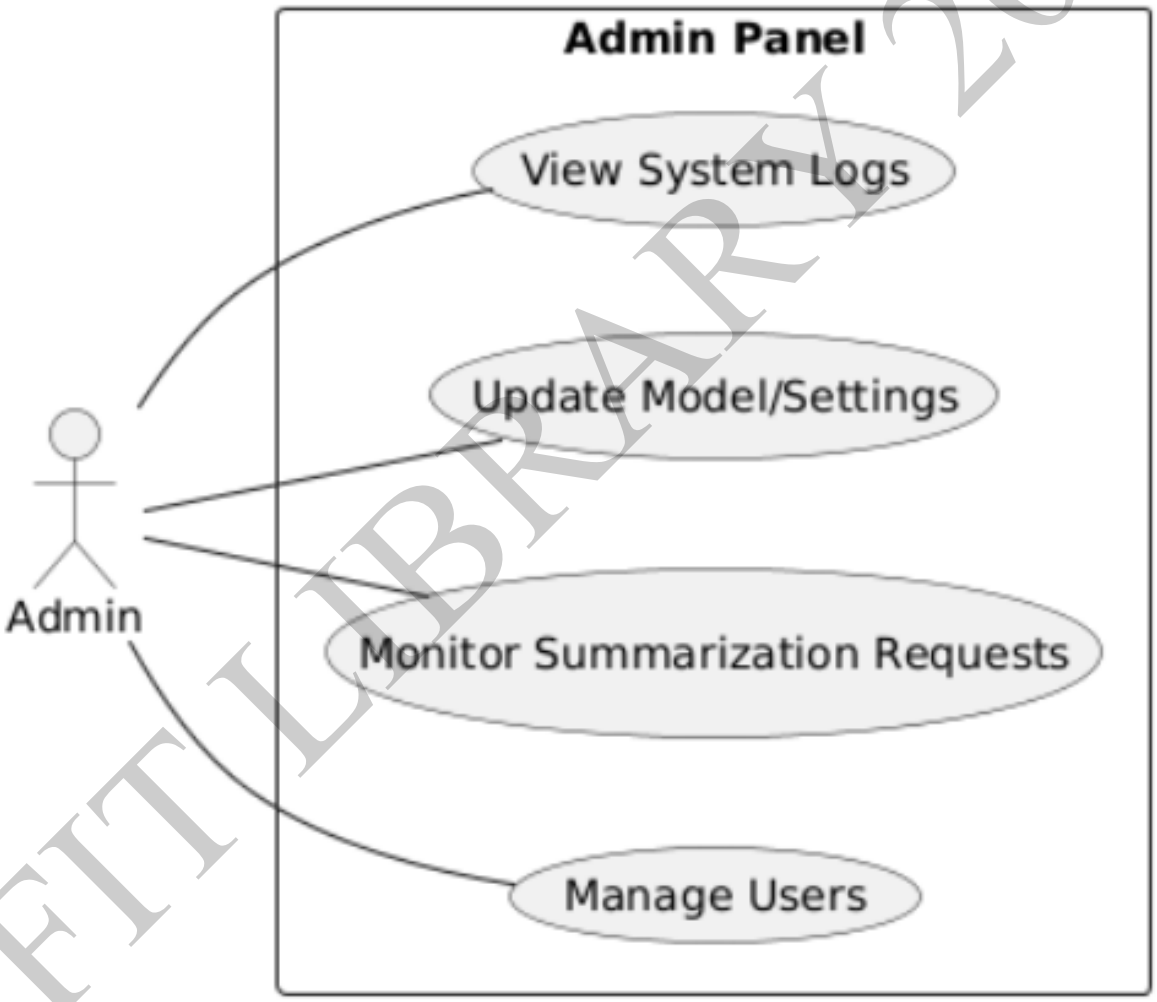

Figure 3.3: Use Case Diagram for System Administrator

The use case diagram highlights the administrator's key responsibilities in managing the system, including user control, model updates, and evaluation processes.

- **Manage Users:** Admin manages user accounts (create, update, delete).

SFIT LIBRARY 2025

- **Monitor Summarization Requests:** Admin monitors all summarization activities and usage statistics.
- **Update Model/Settings:** Admin updates the summarization models or system configurations.
- **Compare Summaries:** User compares the summaries generated by different models.
- **Select Evaluation Metrics:** User chooses metrics (e.g., ROUGE, Recall, Precision) to evaluate the summaries.
- **Evaluate Summary:** User evaluates the generated summaries based on the selected metrics.

SFIT LIBRARY 2025

3.3 Proposed System: Analysis and Modeling

3.3.1 Activity Diagram

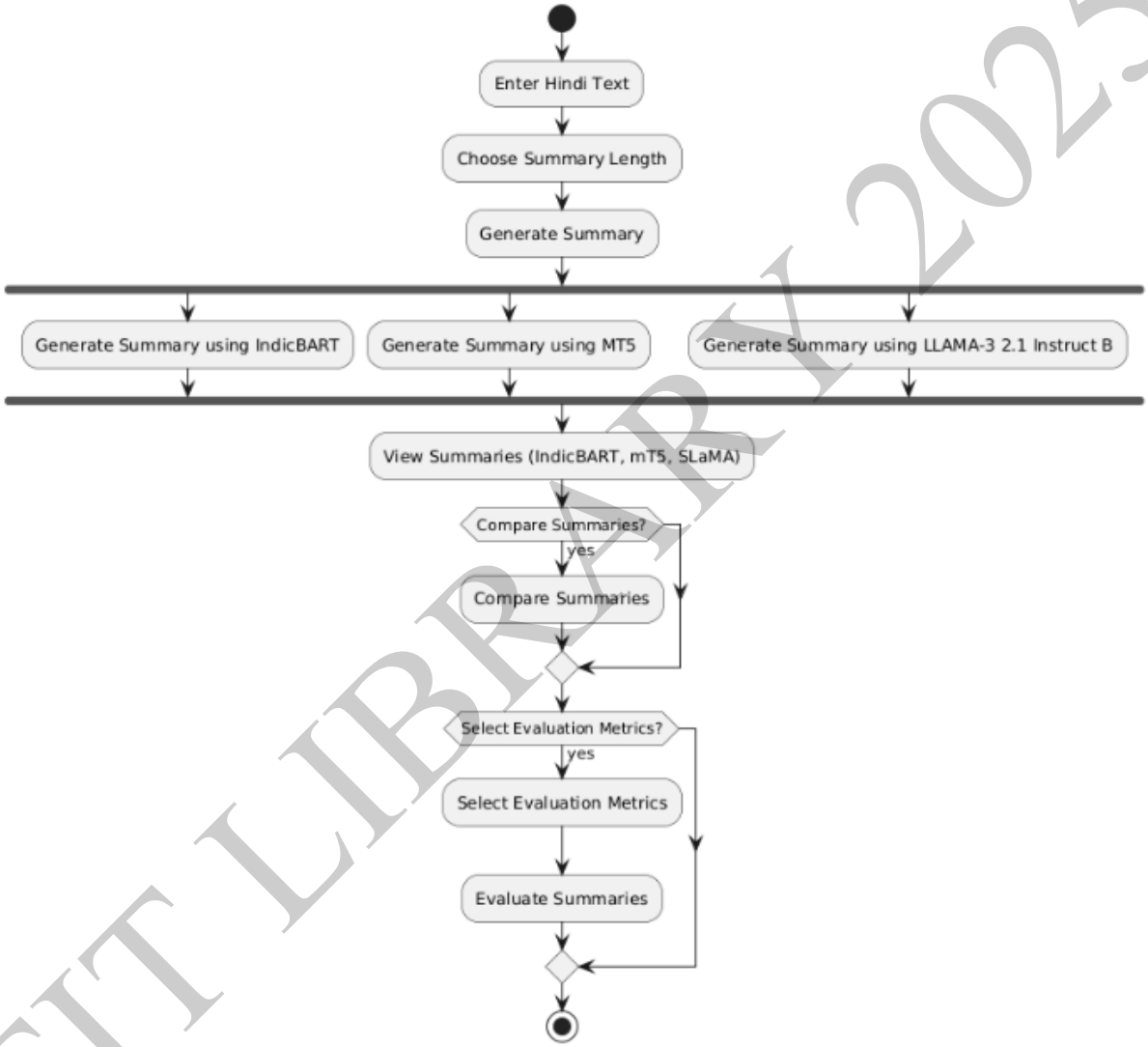

Figure 3.4: Activity Diagarm Of Hindi Text Summarization using LLAMA Models

The activity diagram depicts the workflow of the Hindi Text Summarization system using LLMs like IndicBART, mT5, and LLaMA-3 2.1 Instruct B. It starts when the user inputs Hindi text and selects the desired summary length. The system then concurrently generates summaries using all three models. Users can view each summary individually and compare them side by side to assess differences in output quality. Additionally, the system allows users to select

SFIT LIBRARY 2025

evaluation metrics such as F1 Score, Precision, and Recall. These metrics help in assessing the summarization quality. The workflow is designed for ease of use, letting users generate, compare, and evaluate summaries. It provides insights into how each model performs on the same input. The system ensures structured evaluation. Overall, it offers an efficient approach to multilingual summarization..

3.3.2 Sequence Diagram

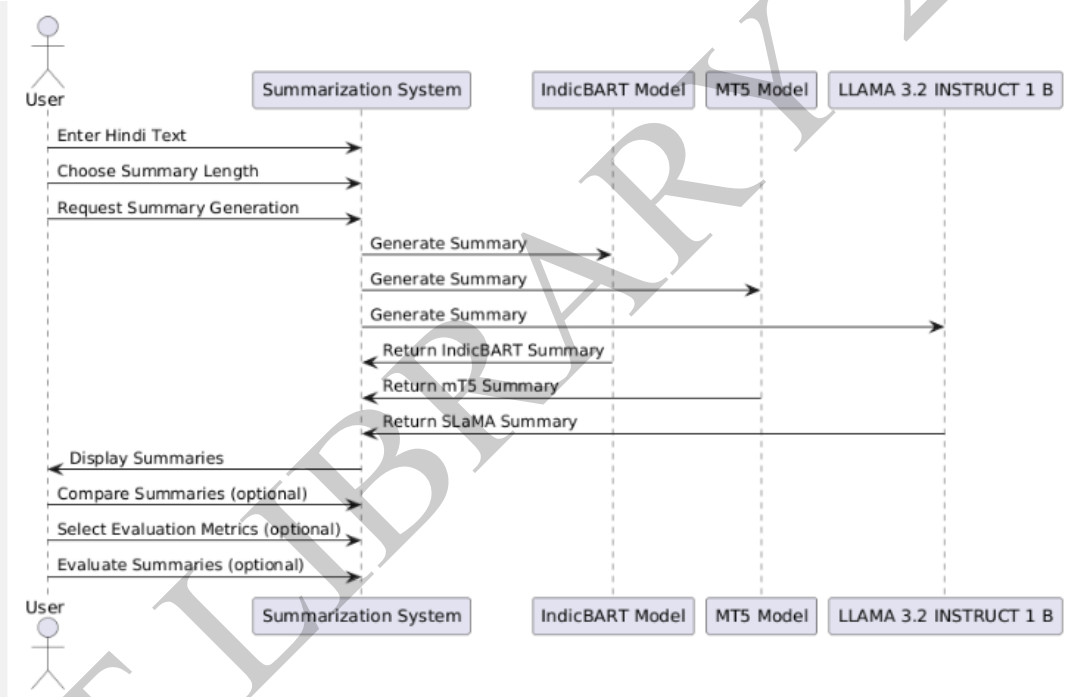

Figure 3.5: Sequence Diagram of Hindi Text Summarization Unsing LLMS Models

The sequence diagram illustrates the detailed interaction flow of the Hindi Text Summarization system using large language models such as IndicBART, mT5, and LLAMA 3 2.1 Instruct B. The user begins by entering the Hindi text into the system and selecting the desired summary length. Upon requesting summary generation, the system sends parallel requests to all three summarization models: IndicBART, mT5, and LLAMA 3.2.1 Instruct B

Each model processes the input text independently and returns its generated summary to the system. Once the system receives the summaries from all mod-

SFIT LIBRARY 2025

els, it displays the results to the user. The user can optionally perform additional actions, such as comparing the summaries to analyze differences and similarities, selecting evaluation metrics like Precision, F1 Score , Recall and evaluating the quality of the summaries based on the chosen metrics.
This sequence captures the real-time interactions between the user, system, and summarization models, ensuring a smooth workflow for generating, viewing, comparing, and evaluating Hindi text summaries.

3.3.3 DFD Diagram

DFD Level 0

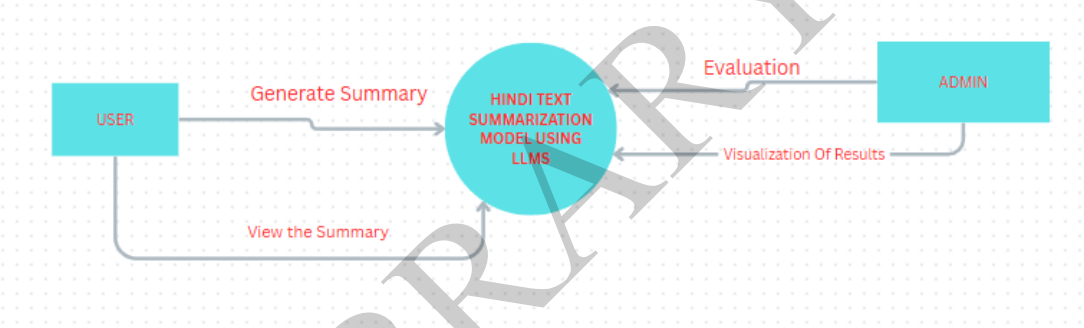

Figure 3.6: DFD Level 0: Overall System Overview

This diagram illustrates the flow of data within a Hindi text summarization system that utilizes LLMs. The User provides Hindi text as input to the system. The system processes this input using LLMs to generate a summary. The User then views the generated summary. Additionally, an Admin interacts with the system to perform evaluations and visualize the results. The diagram shows the key processes and the flow of information between the User, the system, and the Admin.

SFIT LIBRARY 2025

DFD Level 1 - User Interaction

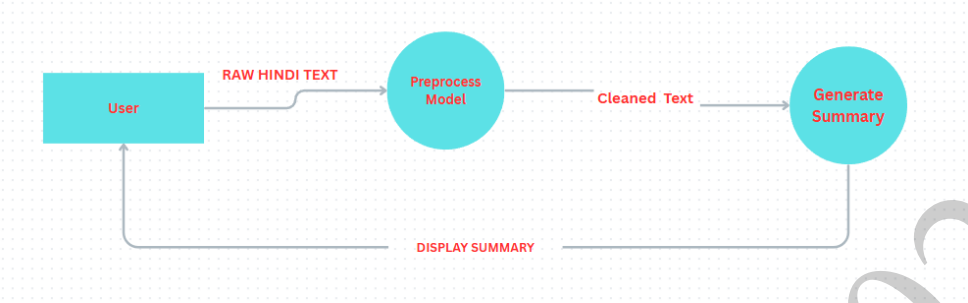

Figure 3.7: DFD Level 1: User Interaction

This Data Flow Diagram (DFD) Level 1 illustrates the user's interaction with the Hindi text summarization system. The user provides raw Hindi text, which is cleaned and preprocessed by removing noise, tokenizing, and normalizing the input. The cleaned text is then processed by a summarization model to generate a concise summary. This summary is finally displayed to the user. The diagram breaks down each interaction step in detail.

DFD Level 2 - User Interaction

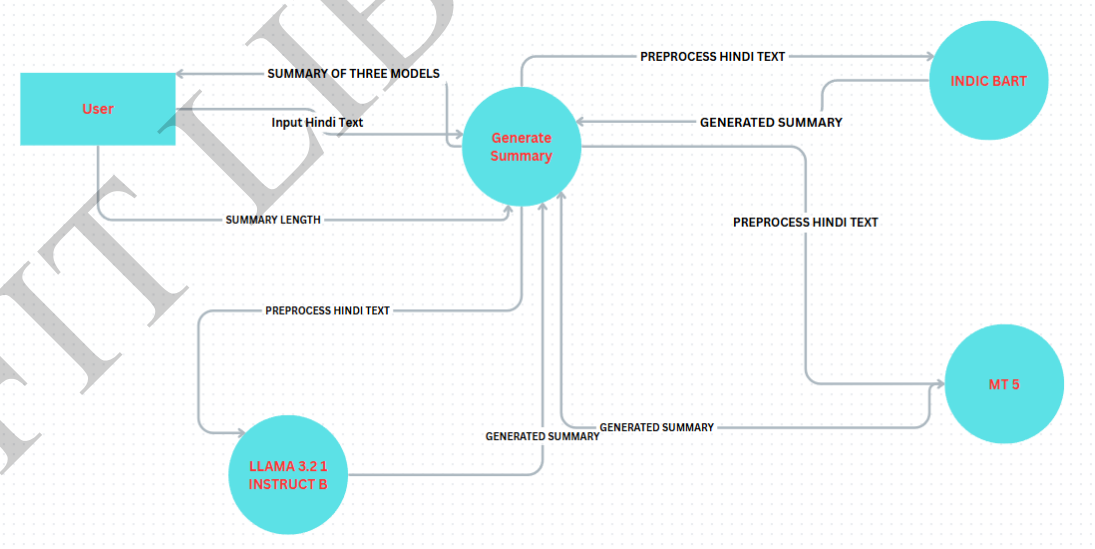

Figure 3.8: DFD Level 2: Detailed User Interaction

This Level 2 DFD refines the "Generate Summary" process by showing how the system uses Indic BART, MT5, and LLaMA 3.2.1 Instruct B to produce individual summaries. Each LLM receives its own preprocessed version of the

SFIT LIBRARY 2025

input Hindi text. The summaries are generated separately and their lengths are computed. The results are then combined and presented to the user. This ensures detailed, multi-model summarization and output clarity.

DFD Level 1 - Admin Interaction

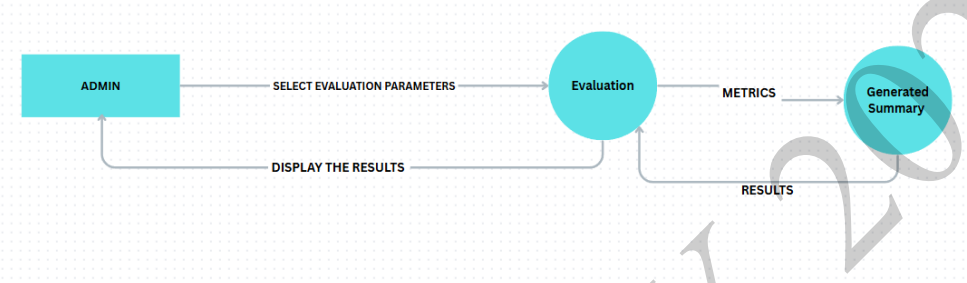

Figure 3.9: DFD Level 1: Admin Interaction

This DFD Level 1 outlines the admin's role in evaluating the summarization system by selecting metrics like BERT Score or F1 Score. The evaluation process compares generated summaries with reference ones using these metrics. It then computes and compiles the results into meaningful evaluation outputs. These results are displayed to the admin. The flow helps assess the system's summarization quality effectively.

SFIT LIBRARY 2025

DFD Level 2 - Admin Interaction

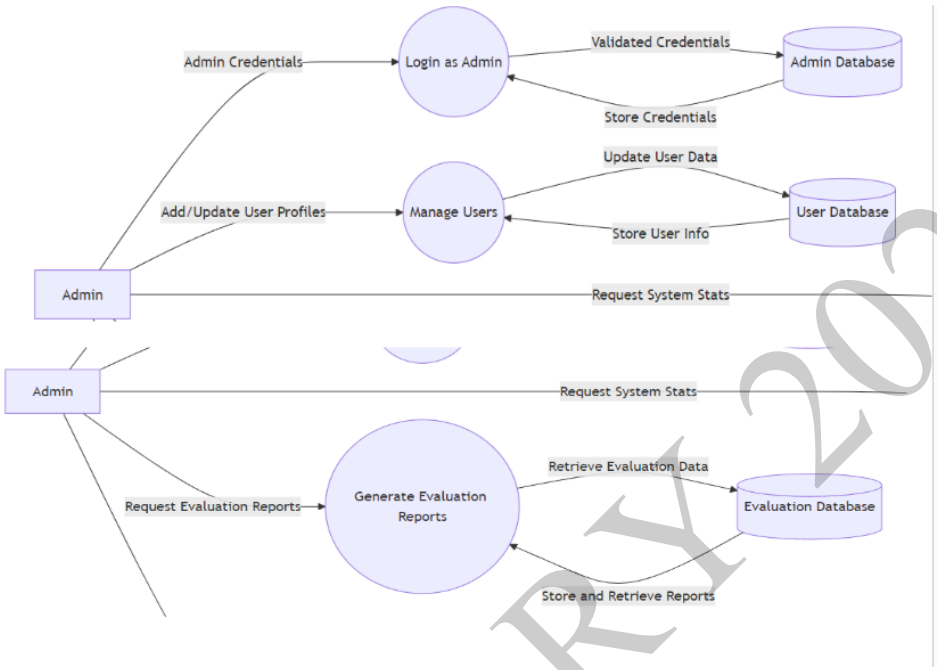

Figure 3.10: DFD Level 2: Admin Interaction

The Level 2 Data Flow Diagram (DFD) for admin interactions in your Hindi Text Summarization System provides a detailed breakdown of the specific processes managed by the admin. The admin initiates the process by logging in, validated against the Admin Database. The admin can then perform actions like generating evaluation reports, which involves retrieving data from the Evaluation Database and storing the reports back. Additionally, the admin uploads or updates summarization models, storing these models in the System Database. The admin also monitors the system's performance by retrieving and storing performance data. This detailed diagram emphasizes the admin's roles in maintaining system functionality, overseeing model updates, and ensuring the overall performance and reliability of the summarization system.

SET LIBRARY 2025

3.3.4 Architectural View

The architecture of Indic BART MT5 and LLAMA 3.2.1 Instruct B models follows a transformer-based structure, consisting of both an encoder and a decoder. These models excel in sequence-to-sequence tasks, leveraging their encoder-decoder layers to process input sequences and generate output sequences efficiently.

Architecture of Indic BART

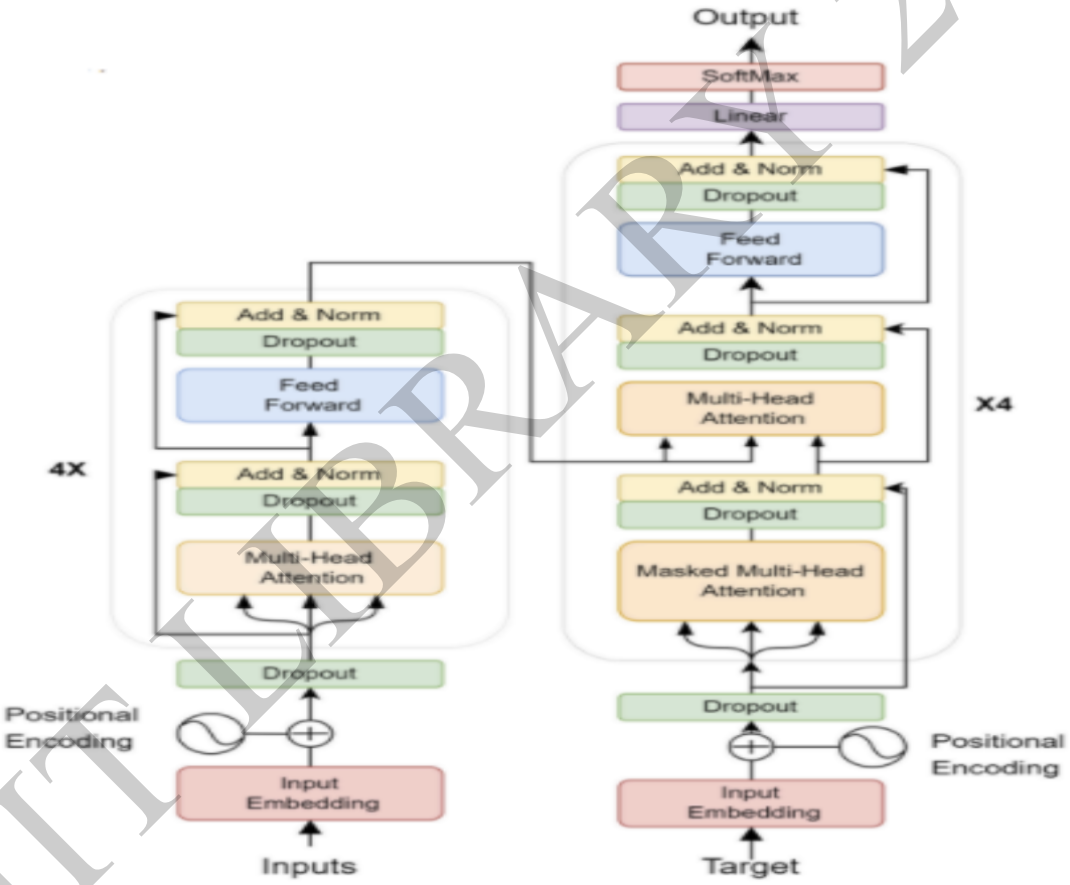

Figure 3.11: Architecture of Indic BART

Encoder

- Input Embedding: Converts the input text into dense vector representations (embeddings).
- Positional Encoding: Adds positional information to the embeddings to maintain the order of words.

SFIT LIBRARY 2025

- Dropout: Applies dropout to prevent overfitting by randomly setting a fraction of input units to zero during training.
- Multi-Head Attention: Uses multiple attention heads to focus on different parts of the input sequence simultaneously.
- Add & Norm: Adds the output of the multi-head attention to the original input (residual connection) and normalizes it.
- Dropout: Applies another dropout layer.
- Feed Forward: Passes the normalized output through a feed-forward neural network.
- Add & Norm: Adds the output of the feed-forward network to its input (residual connection) and normalizes it.
- Dropout: Applies another dropout layer.
- Repeat: Steps 4 to 9 are repeated multiple times (typically 4 times).

Decoder

- Input Embedding: Converts the target text into embeddings.
- Positional Encoding: Adds positional information to the embeddings.
- Dropout: Applies dropout.
- Masked Multi-Head Attention: Uses masked multi-head attention to prevent the model from attending to future tokens in the sequence.
- Add & Norm: Adds the output of the masked multi-head attention to the original input (residual connection) and normalizes it.
- Dropout: Applies another dropout layer.
- Multi-Head Attention: Attends to the encoder's output.

SIT LIBRARY 2025

- Add & Norm: Adds the output of the multi-head attention to its input (residual connection) and normalizes it.
- Dropout: Applies another dropout layer.
- Feed Forward: Passes the normalized output through a feed-forward neural network.
- Add & Norm: Adds the output of the feed-forward network to its input (residual connection) and normalizes it.
- Dropout: Applies another dropout layer.
- Repeat: Steps 4 to 12 are repeated multiple times (typically 4 times).
- Linear: The final output is passed through a linear layer.
- SoftMax: Applies a softmax function to convert the output into probabilities.

SFIT LIBRARY 2025

Key Formulas and Equations for IndicBART Here are some of the essential formulas and equations that underpin the IndicBART model, drawing from the transformer architecture it's based on:

Layer	**Formula**	**Description**
1. Input Embedding	$e_i = Ex_i$	Converts tokens to vectors using embedding matrix
Positional Encoding	$p_{i,2j} = \sin\left(\frac{10000^{j/d_{\text{model}}}}{}\right)$ $p_{i,2j+1} = \cos\left(\frac{10000^{j/d_{\text{model}}}}{}\right)$	Adds position-specific information to the embeddings
Combined Input	$h_i = e_i + p_i$	Combines input embedding and positional encoding
Scaled Dot-Product Attention	$Attention(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$	The attention mechanism that computes weighted sum of values
Multi-Head Attention	$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$ $MultiHead(Q,K,V) = \text{Concat}(head_1,\ldots,head_h)W^O$	Allows the model to jointly attend to information from different representation subspaces
Feed-Forward Network	$FFN(x) = \text{ReLU}(xW_1+b_1)W_2+b_2$	A fully connected feed-forward neural network applied to each position
Add & Norm	$LayerNorm(x+SubLayer(x))$	A residual connection followed by layer normalization
Layer Normalization	$y_i = \frac{x_i-\mu}{\sqrt{\sigma^2+\varepsilon}} \cdot \gamma + \beta$	Formula for layer normalization
Softmax	$P(y\|x) = \text{softmax}(z)$	The Softmax function for probability distribution

Table 3.2: Key Formulas and Equations for IndicBART

SFIT LIBRARY 2025

Architecture of mT5

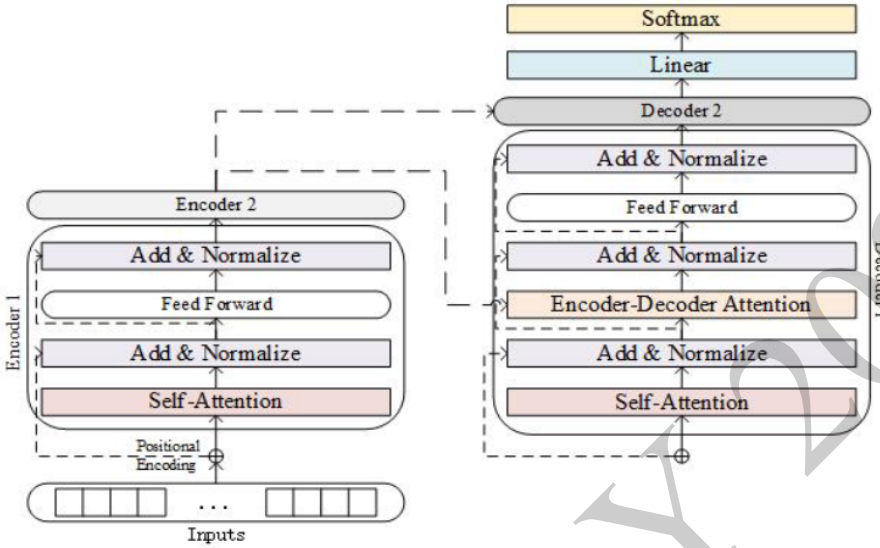

Figure 3.12: Architecture of mT5

Encoder

- Input Embedding: Converts the input text into dense vector representations (embeddings).
- Positional Encoding: Adds positional information to the embeddings to maintain the order of words.
- Dropout: Applies dropout to prevent overfitting by randomly setting a fraction of input units to zero during training.
- Multi-Head Attention: Uses multiple attention heads to focus on different parts of the input sequence simultaneously.
- Add & Norm: Adds the output of the multi-head attention to the original input (residual connection) and normalizes it.
- Dropout: Applies another dropout layer.
- Feed Forward: Passes the normalized output through a feed-forward neural network.

SFIT LIBRARY 2025

- Add & Norm: Adds the output of the feed-forward network to its input (residual connection) and normalizes it.
- Dropout: Applies another dropout layer.
- Repeat: Steps 4 to 9 are repeated multiple times (typically 4 times).

Decoder

- Input Embedding: Converts the target text into embeddings.
- Positional Encoding: Adds positional information to the embeddings.
- Dropout: Applies dropout.
- Masked Multi-Head Attention: Uses masked multi-head attention to prevent the model from attending to future tokens in the sequence.
- Add & Norm: Adds the output of the masked multi-head attention to the original input (residual connection) and normalizes it.
- Dropout: Applies another dropout layer.
- Multi-Head Attention: Attends to the encoder's output.
- Add & Norm: Adds the output of the multi-head attention to its input (residual connection) and normalizes it.
- Dropout: Applies another dropout layer.
- Feed Forward: Passes the normalized output through a feed-forward neural network.
- Add & Norm: Adds the output of the feed-forward network to its input (residual connection) and normalizes it.
- Dropout: Applies another dropout layer.
- Repeat: Steps 4 to 12 are repeated multiple times (typically 4 times).
- Linear: The final output is passed through a linear layer.

SFIT LIBRARY 2025

- SoftMax: Applies a softmax function to convert the output into probabilities.

Key Formulas and Equations for mT5

Component	**Formula**	**Description**
Scaled Dot-Product Attention	$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$	Computes attention weights and applies them to the value vectors.
Feed-Forward Network (FFN)	$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$	Applies two linear transformations with ReLU activation in between.
Positional Encoding	$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$ $PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$	Provides information about the order of the tokens in a sequence.
Layer Normalization	$\text{LayerNorm}(x) = \frac{x-\mu}{\sqrt{\sigma^2+\varepsilon}} \cdot \gamma + \beta$	Normalizes the input to stabilize and accelerate training.
Cross-Entropy Loss	$\mathscr{L} = -\sum_i y_i \log(\hat{y}_i)$	Measures the error between predicted probabilities and true labels.

SFIT LIBRARY 2025

Architecture of LLaMA

LLaMA 3 is built with multiple transformer blocks, advanced attention mechanisms, and efficient normalization techniques.

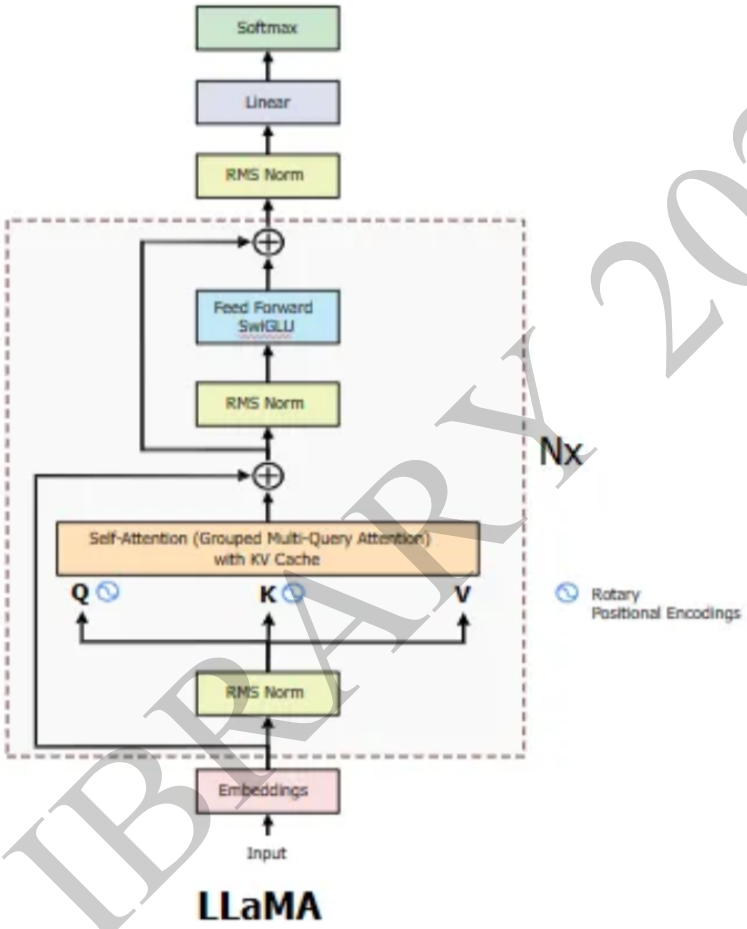

Figure 3.13: Architecture of LLaMA

Input and Token Embeddings

- Converts input text into dense vectors using a learned embedding layer.
- Captures semantic and syntactic meaning of tokens.

Rotary Positional Encoding (RoPE)

- Uses RoPE instead of sinusoidal encoding for better positional awareness.
- Enhances long-context understanding via relative positions in attention.

RMSNorm (Root Mean Square Normalization)

- Replaces LayerNorm to reduce computation and stabilize training.

SFIT LIBRARY 2025

Self-Attention Mechanism (Grouped Multi-Query Attention with KV Cache)

- Employs Grouped MQA where multiple queries share a KV cache.
- KV cache speeds up inference by storing previous key-value pairs.

Feed-Forward Network (SwiGLU Activation)

- Applies FFN with SwiGLU activation to model complex patterns.
- Outperforms traditional ReLU in large-scale models.

Residual Connections and Normalization

- Uses residual connections with RMSNorm to aid gradient flow and stability.

Multiple Transformer Blocks (Nx Layers)

- Contains Nx stacked blocks, each with:
  - Multi-Query Attention (MQA)
  - Feed-Forward Network (SwiGLU)
  - Normalization Layers
  - Residual Connections

Final Layer & Output Softmax

- Final linear layer maps hidden states to logits.
- Softmax converts logits to token probabilities.

SFIT LIBRARY 2025

Formulas Used in LLaMA

Component	Formula and Description
Self-Attention with RoPE	$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{\text{RoPE}(Q)\,\text{RoPE}(K)^\top}{\sqrt{d_k}}\right)V$ RoPE applies rotation-based positional encoding directly to queries Q and keys K.
Grouped Query Attention (GQA)	$\text{Attention}_i(Q,K,V) = \text{softmax}\left(\frac{QW_i^Q(KW_i^K)^\top}{\sqrt{d_k}}\right)VW^V_{\text{group}(i)}$ Multiple heads share keys and values to reduce memory and compute.
Feed-Forward Network with SwiGLU	$\text{FFN}(x) = (\text{SiLU}(xW_1)\odot(xW_2))W_3$ $\text{SiLU}(x) = x\cdot\sigma(x)$ SwiGLU applies a gated nonlinearity using Sigmoid Linear Unit activation.
RMS Normalization	$\text{RMSNorm}(x) = \frac{x}{\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2+\varepsilon}}\times\gamma$ Normalizes based on the root mean square with learned scale γ.
Token Embedding	$x_0 = \text{Embedding}(\text{input})$ Maps input tokens into dense vector representations.
Final Output Projection	$\hat{y} = \text{softmax}(xW_{\text{out}} + b_{\text{out}})$ Transforms final hidden states into output probabilities.

Table 3.4: Formulas Used in LLaMA Architecture

SFIT LIBRARY 2025

3.3.5 Algorithms / Methodology

Data Preprocessing Algorithms

Emoji Removal/Conversion Algorithm:

- Use the `emoji.demojize()` function to convert emojis to their textual descriptions.
- Optionally, use regular expressions to remove any remaining emoji placeholders.

Logic: Normalize the text by handling non-standard characters.

Number to Words Conversion Algorithm:

- Use the `indic-num2words` library to convert numerical values into their corresponding Hindi word representations.
- Apply a regular expression to identify numerical patterns within the text.

Logic: Standardize numerical data for better model understanding.

Text Cleaning (Regular Expressions) Algorithm

- Use `re.sub()` to replace or remove specific patterns in the text.
- Patterns include:
  - Multiple spaces (`\s{2,}`)
  - Special characters (`[<>()| &©@ø`

    ´¨,;:.‘’])
  - Newline characters, tab characters
  - Optionally, remove English characters

Logic: Eliminate noise and inconsistencies in the text.

Lowercasing Algorithm:

- Use the `.lower()` method to convert all characters to lowercase.

Logic: Normalize text to reduce variations.

Tokenization Algorithm:

- Use the tokenizer from the `transformers` library (e.g., `tokenizer_t5`).
- The tokenizer uses subword tokenization (e.g., SentencePiece) to break text into smaller units.

Logic: Prepare text for input to the transformer models.

SFIT LIBRARY 2025

Model Fine-Tuning Methodology

Transformer Architecture (BART/mT5)

Logic: Use a sequence-to-sequence transformer model with an encoder-decoder structure.

- BART: Denoising autoencoder.
- mT5: Multilingual text-to-text transformer.

Fine-Tuning Process Algorithm:

1. Load the pre-trained model and tokenizer.
2. Create a custom Dataset class to prepare the data.
3. Define `TrainingArguments` to configure the training process.
4. Use the `Trainer` class to manage the training loop.
5. The `Trainer` performs the following steps:
    - Forward pass: Input text is passed through the model.
    - Loss calculation: The cross-entropy loss between the predicted and actual summaries is calculated.
    - Backward pass: Gradients are calculated.
    - Parameter updates: Model weights are updated using an optimizer (e.g., AdamW).

Logic: Adapt the pre-trained model to the specific summarization task by updating its weights based on the training data.

Loss Function Equation:

$$L = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{3.1}$$

Logic: Measure the difference between the predicted and actual token distributions.

Optimization Algorithm: AdamW optimizer -
Logic: Efficiently update model parameters based on gradients.

Learning Rate Scheduling Algorithm: Linear or cosine decay -
Logic: Adjust the learning rate during training to improve convergence.

Summary Generation Algorithms

Decoding Algorithm: Beam search -
Logic: Explore multiple possible summaries to find the most likely one.

SFIT LIBRARY 2025

Equation: Beam search maximizes the probability of the generated summary:

$$\hat{Y} = \arg\max_{Y} \prod_{t=1}^{T} P(y_t | y_1, \ldots, y_{t-1}) \tag{3.2}$$

Generation Parameters

- `num_beams`: Number of beams to explore.
- `max_length`: Maximum summary length.
- `min_length`: Minimum summary length.
- `early_stopping`: Stop generation when a sufficient summary is generated.
- `no_repeat_ngram_size`: Prevent repeated n-grams.

Logic: Control the generation process to produce high-quality summaries.

3.3.6 UI/UX DESIGN

The frontend of the Hindi Text Summarizer was developed using Streamlit to create a clean, responsive, and interactive user interface. The main features of the UI are as follows:

1. A large text box for users to easily input Hindi text.
2. A slider to adjust the summary length (from 10% to 60% of the input text).
3. Option to select length priority:
   (a) ”Complete Sentences” to preserve sentence structure.
   (b) ”Exact Length” to meet a strict word count.
4. A ”Generate Summaries” button that produces summaries using three different models:
   (a) IndicBART (specialized for Indian languages).
   (b) mT5 (multilingual model for news summarization).
   (c) Llama 3.2 1B Instruct (fine-tuned for Hindi text summarization).
5. The output displays:
   (a) Summaries generated from each model separately.
   (b) Word counts compared with the target length.
6. The UI uses a dark theme with bright fonts for better readability and a professional look.
7. The design is focused on:
   (a) Simplicity.

SFIT LIBRARY 2025

(b) Ease of use.

(c) Fast interaction.

(d) Clear visualization of results.

The following figures show the UI screens:

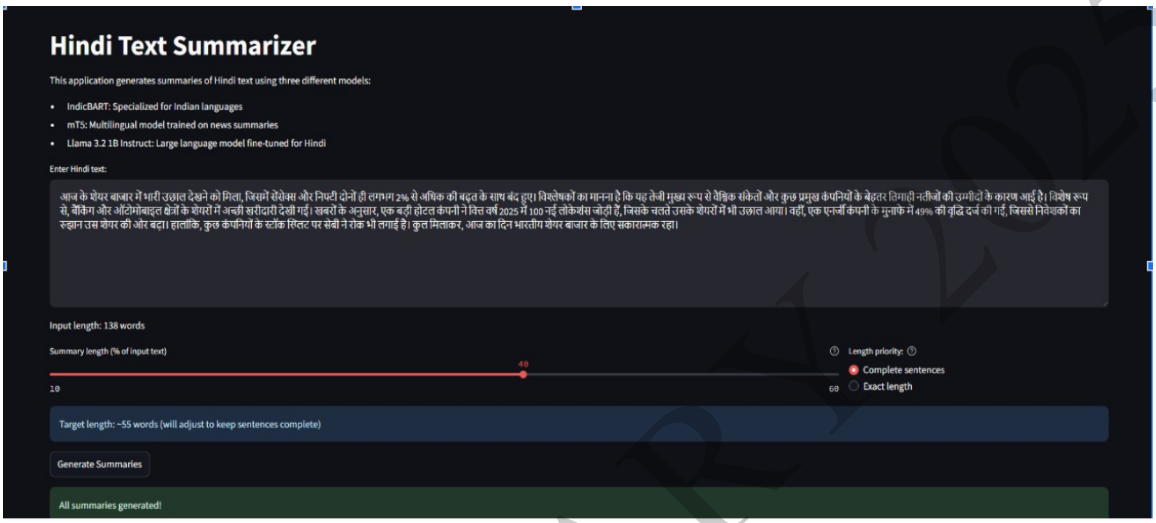

Figure 3.14: Hindi Text Summarizer - Input Text and Summary Settings Screen

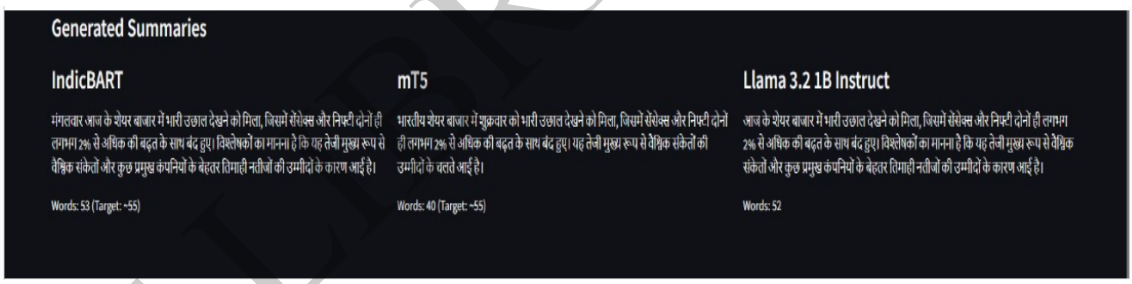

Figure 3.15: Generated Summaries Display Screen (IndicBART, mT5, Llama 3.2 1B)

SFIT LIBRARY 2025

Chapter 4

Implementation Plan and Experimental Set up of the Proposed system

4.1 Experimental Set up

4.1.1 Details discussion of input/Dataset (along with screenshot/ Samples) details about input to systems or selected data

Dataset Description

	Heading	Summary	Article
0	गला दबाकर हत्या की; बॉडी बोरे में भरी, लोकल मा...	Kerala Minor Girl Rape Case - केरल के एर्नाकुल...	केरल के एर्नाकुलम जिले में 5 साल की बच्ची से र...
1	तेलंगाना में 18 की जान गई; जम्मू-कश्मीर में बा...	इस साल मानसून सीजन में कई राज्यों में भारी तबा...	मानसून सीजन में हुई भारी बारिश ने कई राज्यों म...
2	राजस्थान सरकार बनाएगी कर्ज राहत आयोग, कोर्ट के...	चुनावी साल में राजस्थान सरकार किसानों को लुभान...	चुनावी साल में राजस्थान सरकार किसानों को लुभान...
3	3 से 7 अगस्त तक कर सकेंगे अप्लाय, प्राइस बैंड ...	Non-banking lender SBFC Finance's initial publ...	नॉन बैंकिंग फाइनेंस कंपनी 'SBFC फाइनेंस लिमिटेड...
4	डाइनिंग टेबल पर कुकर-कड़ाही न सजाएं, चीन के खा...	स्वाद खाने की बुनियाद है। लेकिन अगर खाना सुंदर...	स्वाद खाने की बुनियाद है। लेकिन अगर खाना सुंदर...

Figure 4.1: Dataset Overview

The dataset used for this research consists of textual data from Hindi political articles and is split into two parts: `train.csv` and `test.csv`. The training dataset contains 2,843 rows, while the test dataset contains 3,001 rows. Each entry in both datasets includes:

- **id**: A unique identifier assigned to each article.
- **Article**: The complete content of the news article in Hindi.
- **Heading**: The headline summarizing the article's key topic.
- **Summary**: A concise summary of the article's content.

SETT LIBRARY 2025

This dataset is employed for training and evaluating the model's ability to generate accurate text summarizations.

Analysis of Dataset

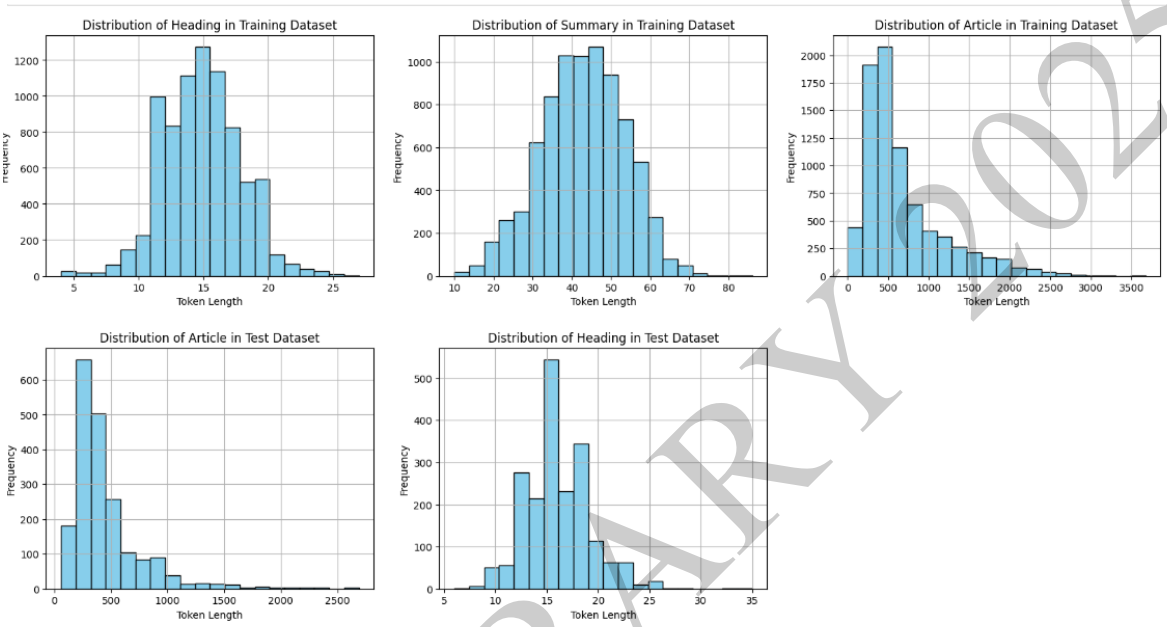

Figure 4.2: Dataset Overview

The dataset analysis provides insights into the token length distributions of various text components in the training and test datasets.

Distribution of Headings in Training Dataset: The histogram shows the distribution of token lengths for headings in the training dataset.

- The most frequent token length for headings is around 15 tokens.
- The distribution is roughly normal, with most headings having between 10 and 20 tokens.

Distribution of Summaries in Training Dataset: The histogram shows the distribution of token lengths for summaries in the training dataset.

- The most frequent token length for summaries is around 45 tokens.
- The distribution is roughly normal, with most summaries having between 30 and 60 tokens.

Distribution of Articles in Training Dataset: The histogram shows the distribution of token lengths for articles in the training dataset.

- The most frequent token length for articles is around 300 tokens.

SFIT LIBRARY 2025

- The distribution is right-skewed, with a long tail extending towards higher token lengths, indicating that while most articles are shorter, there are some very long articles.

Distribution of Articles in Test Dataset: The histogram shows the distribution of token lengths for articles in the test dataset.

- The most frequent token length for articles is around 300 tokens.
- The distribution is right-skewed, similar to the training dataset, with a long tail extending towards higher token lengths.

Distribution of Headings in Test Dataset: The histogram shows the distribution of token lengths for headings in the test dataset.

- The most frequent token length for headings is around 15 tokens.
- The distribution is roughly normal, similar to the training dataset, with most headings having between 10 and 20 tokens.

SFIT LIBRARY 2025

4.1.2 Performance Evaluation Parameters

BERT Precision Score

BERT Precision is a metric that evaluates the accuracy of the summaries generated by models in terms of how many relevant words or phrases are present in the generated summary compared to the reference summary. In the context of Hindi text summarization, it provides insights into the model's ability to produce concise and relevant outputs that align closely with human-generated summaries.
The formula for BERT Precision can be expressed as:

$$\text{BERT Precision} = \frac{\text{No. of Relevant Tokens in Gen.Summary}}{\text{Total No. of Tokens in Gen.Summary}} \tag{4.1}$$

No. Number

Gen. Generated

BERT Recall Score

BERT Score Recall measures the proportion of relevant words or phrases from the reference summary that are captured in the generated summary. In the context of Hindi text summarization, it assesses how well the model identifies and includes key information from the original text, indicating the completeness of the generated summary.

$$\text{Recall} = \frac{\sum_{w \in \text{ref}} \text{BERT}(w, \text{gen})}{\sum_{w \in \text{ref}} \text{BERT}(w, \text{ref})} \tag{4.2}$$

BERT F1 Score

BERT Score F1 combines both precision and recall into a single metric, providing a balanced measure of the model's performance in generating summaries. In the context of Hindi text summarization, it assesses how well the model produces relevant content while also capturing the essential information from the original text.
BERT Score F1 is calculated using the following formula:

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.3}$$

4.1.3 Special Requirement

In this section, we describe the experimental setup used to evaluate the performance of our models. We employed various software tools and libraries to implement the models and perform Hindi text summarization tasks. Below is a brief description of the setup:

- **Python 3.8+**: Python was the core programming language used to implement the summarization models and handle data processing tasks.

SFIT LIBRARY 2025

- **PyTorch 1.8+**: PyTorch was used as the primary deep learning framework to build and train the models, offering flexibility and efficiency for GPU-based training.
- **Transformers Library**: We utilized the Hugging Face Transformers library, which provided pre-trained models and simplified the fine-tuning process for text summarization tasks.
- **TensorFlow**: TensorFlow was also used for some tasks, especially when comparing the performance of models trained using this framework versus those in PyTorch.
- **Hugging Face**: Hugging Face played a crucial role, offering pre-trained models, datasets, and tools that allowed for quick experimentation and efficient model fine-tuning in Hindi text summarization.

The dataset was prepared by tokenizing and normalizing the Hindi text, followed by splitting it into training, validation, and test sets. Model training and evaluation were performed using hyperparameter tuning strategies such as:

- Learning rate scheduling
- Dropout regularization
- Early stopping

These strategies were applied to ensure optimal performance and avoid overfitting.

Fine Tuning

In our project, we utilized the IndicBART, MT5, and Transformer models for Hindi text summarization, leveraging the Hugging Face Transformers library along with PyTorch for fine-tuning. The fine-tuning process is guided by specific configurations outlined in the config.json and generation- config.json files. The IndicBART model is built on the MBart architecture, designed for conditional generation tasks, and consists of six layers each in the encoder and decoder with 16 attention heads. This structure allows the model to capture complex dependencies in the input data effectively. Key dropout settings are set at 0.1 for both activation and attention, which helps prevent overfitting during training. The feed-forward network is defined with a dimension of 4096, providing sufficient capacity for learning intricate patterns. Additionally, specific token IDs for beginning-of-sequence (BOS) and end-of-sequence (EOS) are configured to facilitate proper sequence handling during generation. Overall, these configurations ensure that the IndicBART model is optimized for generating coherent and context-aware summaries of Hindi text, enhancing the performance and stability of our summa- rization task.

Hugging Face And Pytorch

Hugging Face and PyTorch play pivotal roles in our project, particularly in implementing and fine-tuning the IndicBART, MT5, and Transformer model for Hindi text summarization. Hugging Face provides a user-friendly interface and a vast repository of pre-trained models, simplifying the integration of advanced natural language processing capabilities. Its Trans- formers library facilitates seamless model

SIT LIBRARY 2025

loading, tokeniza- tion, and generation tasks, enabling rapid experimentation and development. On the other hand, PyTorch offers a flexible and efficient deep learning framework, allowing us to customize model architectures and optimize training processes with ease. The combination of Hugging Face's extensive model library and PyTorch's dynamic computational graph empowers us to leverage state-of-the-art techniques in our summarization task, enhancing both performance and efficiency.

SFIT LIBRARY 2025

4.2 Code

4.2.1 Model Implementation

BART Model for Summary Generation

Below is the Python code used to generate summaries using the BART model. This implementation fine-tunes the BART model to summarize Hindi text.

BART Model for Summary Generation

```
model_bart = AutoModelForSeq2SeqLM.from_pretrained(bart_checkpoint).
    to(device)

def generate_summary(article, model):
    inp = tokenizer_bart(article, add_special_tokens=False,
                         truncation=True, return_tensors="pt",
                         padding='max_length', max_length=1024)['
                            input_ids']
    inp = inp.to(device)  # Move input tensor to the desired device

    model_output = model.generate(inp, use_cache=True,
                                  num_beams=4,
                                  max_length=70,
                                  min_length=30,
                                  early_stopping=True,
                                  pad_token_id=pad_id,
                                  bos_token_id=bos_id,
                                  eos_token_id=eos_id,
                                  decoder_start_token_id=
                                     tokenizer_bart.
                                     _convert_token_to_id_with_added_voc
                                     ("<2en>"))

    decoded_output = tokenizer_bart.decode(model_output[0],
                                           skip_special_tokens=True,
                                           clean_up_tokenization_spaces
                                              =False)
    return decoded_output

SystemSummary1 = []
for index, row in tqdm(test_df.iterrows(), total=len(test_df)):
    summary = generate_summary(row['Article'], model_bart)
    SystemSummary1.append(summary)
```

SFIT LIBRARY 2025

Explanation of the Code

The provided Python code implements a text summarization pipeline using the **BART (Bidirectional and Auto-Regressive Transformer)** model.

- **Loading the Model:** The BART model is loaded from a pretrained checkpoint and moved to the appropriate device (CPU/GPU).
- **Tokenization:** The input article is tokenized and padded to a maximum length of 1024 tokens to ensure consistency.
- **Generating the Summary:** The `generate_summary()` function generates a summary using beam search with constraints on the minimum and maximum summary length.
- **Decoding the Output:** The generated token sequence is converted back into a human-readable summary using `tokenizer_bart.decode()`.
- **Processing the Dataset:** The code iterates over the test dataset, applies the `generate_summary()` function, and stores the results in `SystemSummary1`.

This approach enables efficient text summarization while maintaining fluency and relevance in the generated summaries.

Code Implementation of Models and Evaluation

The following code implements the evaluation framework for the Hindi text summarization models (IndicBART, XLSum, and Llama). It includes data loading, model loading, summary generation, and ROUGE score calculation.

Code Implementation of Models and Evaluation

```
import pandas as pd
import torch
import streamlit as st
from rouge_score import rouge_scorer
import os, re, math, sys
from tqdm import tqdm

# Load model (simplified)
def load_model_optimized(model_name):
  #  Load tokenizer and model based on model_name
  pass

# Generate summary (simplified)
def generate_summary(article, tokenizer, model, model_name,
    target_length):
  # Generate summary using the specified model
  pass
```

SIT LIBRARY 2025

```
17
18 def clear_gpu_memory():
19   # Clear GPU memory
20   pass
21
22 def count_hindi_words(text):
23   # Count words in Hindi text
24   pass
25
26 def normalize_text(text):
27   # Normalize the input text
28   text = re.sub(r'\s+', ' ', text).strip()
29   text = re.sub(r'[^\w\s\.\?\!      ,\u0900-\u097F]', '', text)
30   return text.lower()
31
32 # Load test data
33 def load_test_data(file_path="./dataset/test.csv", sample_size=None):
34   try:
35     df = pd.read_csv(file_path, encoding='utf-8', on_bad_lines='skip'
         , quoting=3)
36     df = df.rename(columns={"text": "article", "headline": "summary"
         }) #if needed
37     df = df.dropna(subset=["article", "summary"]).sample(n=
         sample_size) if sample_size else df.dropna(subset=["article",
         "summary"])
38     return df
39   except Exception as e:
40     st.error(f"Error loading data: {e}")
41     return None
42
43 # Calculate ROUGE scores
44 def calculate_rouge_scores(reference, hypothesis):
45   if not reference or not hypothesis:
46     return {"rouge1_f": 0, "rouge2_f": 0, "rougeL_f": 0}
47   reference, hypothesis = normalize_text(reference), normalize_text(
       hypothesis)
48   scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'],
       use_stemmer=False)
49   scores = scorer.score(reference, hypothesis)
50   return {f"rouge{k}_f": v.fmeasure for k, v in scores.items()}
51
52 def calculate_scores(reference, hypothesis):
53   results = calculate_rouge_scores(reference, hypothesis)
54   ref_len, hyp_len = count_hindi_words(reference), count_hindi_words(
       hypothesis)
55   results["length_ratio"] = hyp_len / ref_len if ref_len else 0
56   if ref_len and hyp_len:
```

SELF LIBRARY 2025

```
    ref_freq = {w: reference.split().count(w) for w in set(reference.split())}
    hyp_freq = {w: hypothesis.split().count(w) for w in set(hypothesis.split())}
    all_words = set(ref_freq) | set(hyp_freq)
    vec1 = [ref_freq.get(w, 0) for w in all_words]
    vec2 = [hyp_freq.get(w, 0) for w in all_words]
    cosine_sim = sum(a*b for a,b in zip(vec1, vec2)) / (math.sqrt(sum(a*a for a in vec1)) * math.sqrt(sum(b*b for b in vec2))) if any(vec1) and any(vec2) else 0
    results["cosine_sim"] = cosine_sim
  results["char_overlap"] = len(set(reference.replace(" ", "")) & set(hypothesis.replace(" ", ""))) / len(set(reference.replace(" ", "")) | set(hypothesis.replace(" ", ""))) if reference else 0
  return results

# Evaluate models
def evaluate_models(test_data, target_percentage=30, max_samples=50):
  results = {model: {"rouge1": [], "rouge2": [], "rougeL": [], "length_ratio": [], "char_overlap": [], "cosine_sim": []} for model in ["IndicBART", "XLSum", "Llama"]}
  debug_examples = []
  samples = min(len(test_data), max_samples)
  for i, row in tqdm(test_data.head(samples).iterrows(), total=samples):
    article, reference = row["article"], row["summary"]
    target_length = max(int(count_hindi_words(article) * (target_percentage / 100)), 30)
    for model_name in ["IndicBART", "XLSum", "Llama"]:
      try:
        tokenizer, model = load_model_optimized(model_name)
        hypothesis = generate_summary(article, tokenizer, model, model_name.lower(), target_length)
        if hypothesis and count_hindi_words(hypothesis) > target_length * 1.2:
          hypothesis = smart_truncate(hypothesis, target_length)
        scores = calculate_scores(reference, hypothesis)
        for metric in results[model_name]:
          results[model_name][metric].append(scores.get(metric,0))
        if i == 0:
          debug_examples.append({
            "model": model_name,
            "reference": reference,
            "generated": hypothesis,
            "ref_length": count_hindi_words(reference),
            "gen_length": count_hindi_words(hypothesis),
```

SIT LIBRARY 2025

```
            **{k: scores.get(k, 0) for k in ["rouge1_f", "rouge2_f",
                "rougeL_f", "length_ratio", "char_overlap", "
                cosine_sim"]}
          })
        del model, tokenizer; clear_gpu_memory()
      except Exception as e:
        st.error(f"Error with {model_name}: {e}")
  avg_results = {model: {k.replace('_f','').upper(): sum(v) / len(v)
      if v else 0 for k, v in scores.items()} for model, scores in
      results.items()}
  return avg_results, results, debug_examples

def main():
  st.title("Hindi Summarizer Evaluation")
  test_file = st.text_input("Path to test CSV:", "./dataset/
      sample_test.csv")
  sample_size = st.number_input("Samples:", 1, 100, 10)
  target_percentage = st.slider("Target length (%):", 10, 50, 30)
  use_manual = st.checkbox("Use manual data")

  if st.sidebar.button("Create Sample Data"):
    try:
      pd.DataFrame({
        "article": ["Sample 1", "Sample 2"],
        "summary": ["Summary 1", "Summary 2"]
      }).to_csv('./dataset/sample_test.csv', index=False, encoding='
          utf-8')
      st.sidebar.success("Sample data created at ./dataset/
          sample_test.csv")
    except Exception as e:
      st.sidebar.error(f"Error: {e}")

  if use_manual:
    manual_article = st.text_area("Article:", height=200)
    manual_summary = st.text_area("Reference Summary:", height=100)
    if st.button("Run") and manual_article and manual_summary:
      test_data = pd.DataFrame({"article": [manual_article], "summary
          ": [manual_summary]})
  elif st.button("Run"):
    test_data = load_test_data(test_file, sample_size)

  if test_data is not None:
    avg_results, detailed_results, debug_examples = evaluate_models(
        test_data, target_percentage, sample_size)
    st.subheader("Results")
    st.write(pd.DataFrame(avg_results).T.round(4) * 100)
    for ex in debug_examples:
```

SIT LIBRARY 2025

```
129         st.markdown(f"### Model: {ex['model']}\n**Reference Summary ({
                ex['ref_length']} words):**\n{ex['reference']}\n**Generated
                Summary ({ex['gen_length']} words):**\n{ex['generated']}\n**
                Metrics:**\n" +
130                         ''.join([f"- {k.replace('_f','').upper()}: {v
                                :.4f}\n" for k, v in ex.items() if k in ["
                                rouge1_f", "rouge2_f", "rougeL_f",  "
                                length_ratio", "char_overlap", "cosine_sim"
                                ]]))
```

This code defines functions for loading data, evaluating models (IndicBART, XLSum, Llama), and calculating metrics such as ROUGE scores, length ratio, character overlap, and word similarity. It processes input articles, generates summaries using the specified models, and compares them against reference summaries. The main function orchestrates the evaluation process, allowing users to input data either from a CSV file or manually, and displays the results.

SFIT LIBRARY 2025

Chapter 5

Results and Discussions

5.1 Presentation and validation of the results for the proposed system

Result of IndicBart	
BERTScore Precision	0.690012
BERTScore Recall	0.712243
BERTScore F1	0.704588
Result of MT5	
BERTScore Precision	0.543827
BERTScore Recall	0.554322
BERTScore F1	0.532156
Result of Llama	
BERTScore Precision	0.753459
BERTScore Recall	0.770091
BERTScore F1	0.766667

Figure 5.1: Results of all Evaluation Metrics of all Models

5.2 Comparative Analysis

This section presents a comparative analysis of the proposed system against existing transformer-based models used for Hindi text summarization. The table below provides a detailed comparison of BERT Precision, BERT Recall, and BERT F1 Score across different models.

SFIT LIBRARY 2025

Model	BERT Precision	BERT Recall	BERT F1 Score
Indic BART	0.690012	0.712243	0.704588
mT5	0.543827	0.554322	0.532156
LLaMA 3	**0.753459**	**0.770091**	**0.766667**

Table 5.1: Comparison of BERT Precision, Recall, and F1 Score Across Models

5.2.1 *Model Comparison: BERT Metrics

Analysis

- **Metrics:** Precision, Recall, and F1-score evaluate model accuracy in identifying relevant information.
- **LLaMA 3:** Highest scores across all metrics.
- **Indic BART:** Second-best performance.
- **mT5:** Lowest performance.

Prediction

- **Best Model:** LLaMA 3 (precise and high coverage).
- **Second Best:** Indic BART.
- **Least Effective:** mT5.
- **Summary:** LLaMA 3 is the most effective overall.

SFIT LIBRARY 2025

5.2.2 Graphical Analysis of Indic BART Evaluation

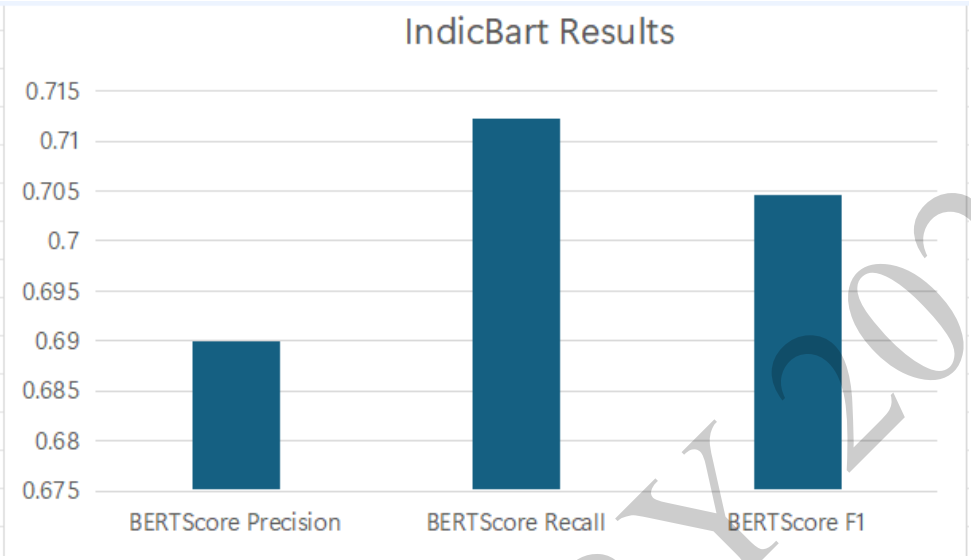

Figure 5.2: Indic BART Evaluation Results

5.2.3 Graphical Analysis of MT5 Evaluation

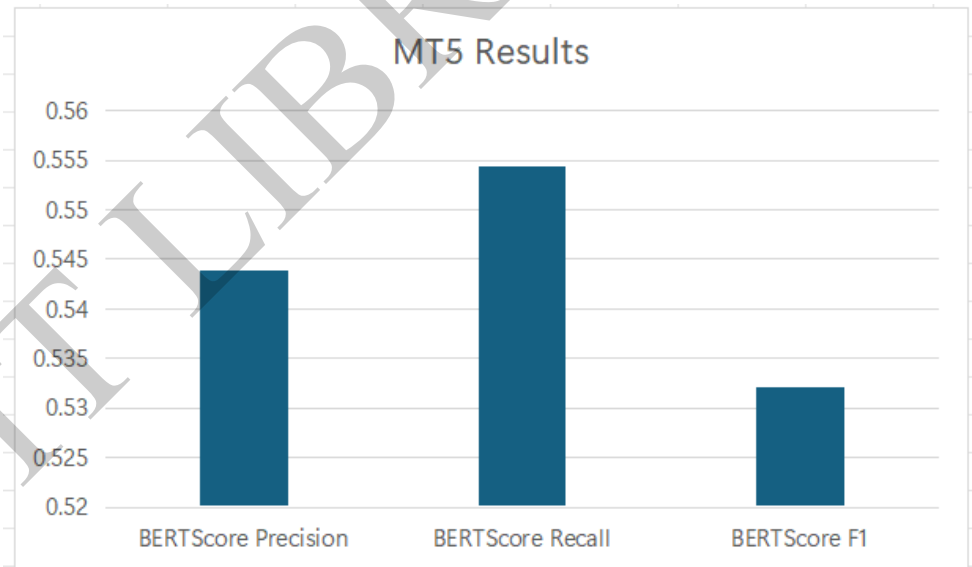

Figure 5.3: MT5 Evaluation Results

SFIT LIBRARY 2025

5.2.4 Graphical Analysis of LLAMA Evaluation

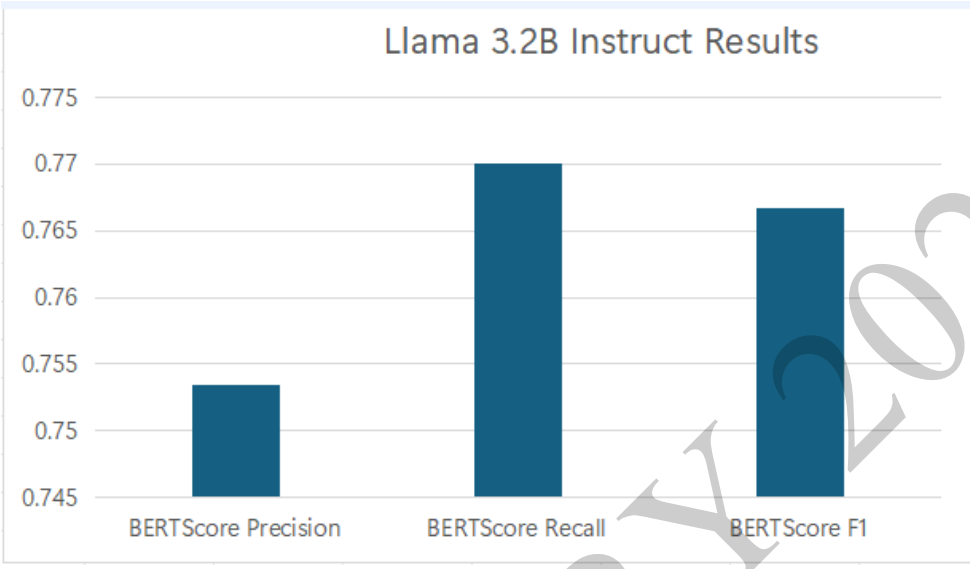

Figure 5.4: LLAMA 3.2.1 Instruct B Evaluation Results

5.2.5 Comparative Analysis Of Evaluation Of All Three Models

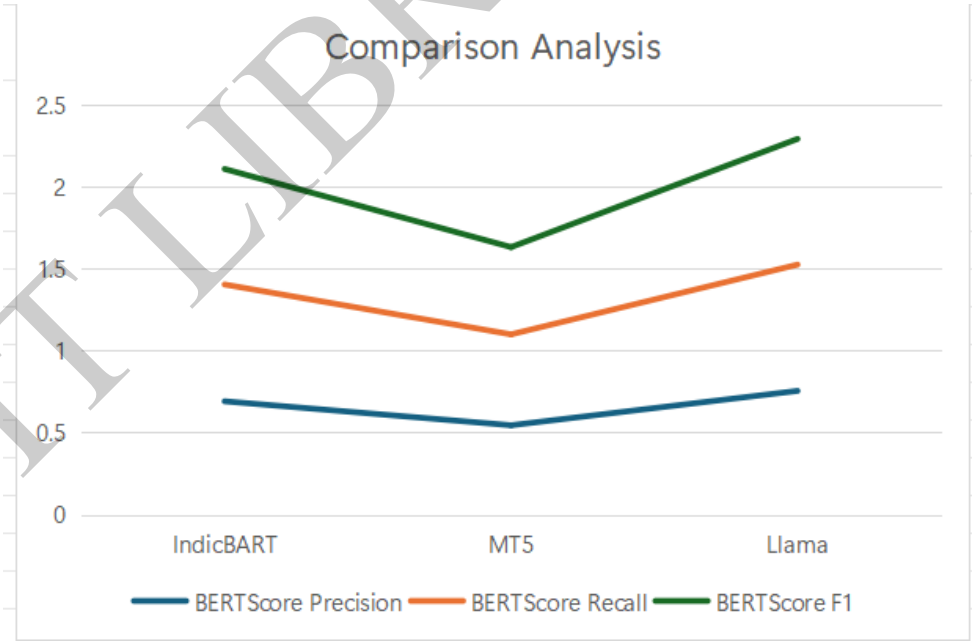

Figure 5.5: Comparative Analysis of All Three Models(Indic BART, MT5 and LLAMA

SFIT LIBRARY 2025

Chapter 6

Conclusion

In this semester, the Hindi text summarization project successfully progressed through key phases, including data preprocessing, model training, and evaluation. Three advanced models—IndicBART, mT5, and LLaMA 3.2.1 Instruct B—were used for generating summaries. Preprocessing involved tokenization, normalization, and dataset splitting into training, validation, and testing sets. All models were fine-tuned on Hindi text data and evaluated using BERT-based metrics: Precision, Recall, and F1 Score. Among the three, LLaMA 3 delivered the best performance with the highest scores across all evaluation metrics. IndicBART ranked second, showing reasonable performance, while mT5 lagged behind in both accuracy and relevance. Specifically, LLaMA 3 achieved an F1 Score of 0.766667, outperforming IndicBART (0.704588) and mT5 (0.532156). The results indicate that LLaMA 3 provides the best balance between precision and coverage for summarization tasks in the Hindi language. Furthermore, the project showcased the importance of fine-tuning transformer-based models on domain-specific datasets for improved performance. Visual inspection of summaries also supported the quantitative findings, with LLaMA 3 generating more coherent and contextually rich summaries. This phase of the project demonstrated a successful implementation pipeline from data preparation to evaluation. It also helped build practical skills in working with NLP tools and models. Overall, the semester's work laid a strong foundation for future exploration, optimization, and integration of summarization models into real-world applications.

SFIT LIBRARY 2025

Appendix-I

We provides additional insights and technical details relevant to the Hindi text summarization project, enriching the understanding of the models, datasets, and methodologies applied. This appendix begins with a comprehensive breakdown of the datasets used, including sources, characteristics, and any language-specific preprocessing steps, such as tokenization, normalization, and stop-word removal. Given the complexity of working with Hindi text, which involves handling unique grammatical structures and vocabulary, these preprocessing steps are crucial for preparing the data effectively for summarization tasks. Detailed information on preprocessing procedures and sample inputs and outputs helps illustrate how data was transformed to optimize performance and maintain linguistic accuracy.

Additionally, Appendix - I delves into the technical configurations of the Indic BART and mT5 models, documenting parameter settings, model architecture, and the fine-tuning process to ensure relevance to Hindi text. This section might also cover the implementation challenges encountered, such as adjusting hyperparameters for better model accuracy, computational resource management, and compatibility with Python libraries. An outline of evaluation metrics, like BERT Precision, Recall, and F1 Score, with detailed explanations and sample calculations, provides a framework to gauge the performance of these models in summarizing Hindi text accurately. Together, this appendix offers a valuable resource for understanding the foundational steps, decisions, and technical nuances involved in setting up and assessing a robust Hindi text summarization model.

SFIT LIBRARY 2025

References

[1] A. Agarwal, S. Naik, and S. Sonawane, "Abstractive Text Summarization for Hindi Language using IndicBART," in *Proceedings of the 2022 IEEE 14th Pacific Visualization Symposium (PacificVis)*, Kolkata, India, 2021, pp. 1–10.

[2] D. Taunk and V. Varma, "Summarizing Indian Languages using Multilingual Transformers based Models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 28, no. 1, pp. 217–226, Jan. 2022.

[3] A. Krishnakumar, F. Naushin A. R., M. K. L., and B. Bharathi, "Text summarization for Indian languages using pre-trained models," *IEEE Access*, vol. 11, pp. 45181–45193, 2023.

[4] R. Mishra, "Automatic Text Summarization of Hindi Text using Real Coded Genetic Algorithm," in *Proceedings of the 2022 ACM Symposium*, 2022.

[5] S. Sharma and N. Aggarwal, "Abstractive Text Summarization Using Seq-ToSeq Neural Network with Attention for Hindi Language," *IEEE Transactions*, 2021.

SFIT LIBRARY 2025

Acknowledgements

We are thankful to a number of individuals who have contributed towards our final year project and without their help; it would not have been possible. First of all, we would like to express our gratitude to **Pradnya Sawant** our project guide for their prompt suggestions, guidance and encouragement over the course of our entire project.

We sincerely appreciate **Pradnya Sawant**, our project coordinators, for their tremendous assistance to our project. We also appreciate the support of the faculty in our department.

We express our sincere gratitude to our respected Director Bro. Shantilal Kujur, our Principal Dr. Sincy George and our Head of Department (CMPN) Dr. Kavita Sonawane for providing the facilities, encouragement as well as a conducive environment for learning.

Lastly, we would like to thank our parents and friends who played a crucial role keeping us motivated throughout with their constant nurture and support.

Sincerely,

Pranay Chavan

Jennessa Rowena Dsouza

Om Kamerkar

Mansi Chaklashiya

SFIT LIBRARY 2025