Text Summarization

A major project report submitted in partial fulfillment of the requirement for the award of degree of

Bachelor of Technology

in

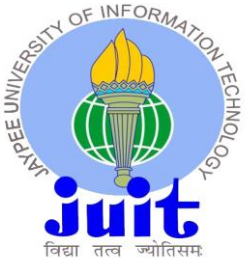
Computer Science & Engineering / Information Technology

Submitted by

Aniket Sharma (201350), Rithik Mishra (201404)

Under the guidance & supervision of

Prof. Dr. Vivek Kumar Sehgal



Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, Solan - 173234 (India)

Certificate

This is to certify that the work which is being presented in the project report titled Text Summarization in partial fulfillment of the requirements for the award of the degree of B.Tech in Computer Science And Engineering and submitted to the Department of Computer Science And Engineering, Jaypee University of Information Technology, Waknaghat is an authentic record of work carried out by Aniket Sharma & Rithik Mishra with Roll Number 201350 & 201404 respectively during the period from August 2023 to May 2024 under the supervision Prof. Dr. Vivek Kumar Sehgal (HOD, Department of Computer Science & Engineering and Information Technology)

Student Name: Aniket Sharma
Roll No.: 201350

Student Name: Rithik Mishra
Roll No.: 201404

The above statement made is correct to the best of my knowledge.

Supervisor Name: Prof. Dr. Vivek Kumar Sehgal
Designation: HOD
Department: Computer Science and Engineering and Information Technology

Candidate's Declaration

I hereby declare that the work presented in this report entitled **Text Summarization** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering / Information Technology** submitted in the Department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology, Waknaghat is an authentic record of my own work carried out over a period from August 2023 to May 2024 under the supervision of **Prof. Dr. Vivek Kumar Sehgal** (HOD, Department of Computer Science & Engineering and Information Technology).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Student Name: Aniket Sharma
Roll No.: 201350

Student Name: Rithik Mishra
Roll No.: 201404

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Supervisor Name: Prof. Dr. Vivek Kumar Sehgal
Designation: HOD
Department:Computer Science and Engineering and Information Technology

Acknowledgement

Firstly, we express our heartiest thanks and gratefulness to almighty God for his divine blessing making it possible to complete the project work successfully.

We are really grateful to our supervisor Prof. Dr. Vivek Kumar Sehgal (HOD, Department of Computer Science & Engineering and Information Technology) to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patients of my parents.

Student Name: Aniket Sharma
Roll No.: 201350

Student Name: Rithik Mishra
Roll No.: 201404

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract

Our end-to-end NLP project on text summarization represents a comprehensive study of the entire development life cycle. The project is built on a Llama, a transformer-based model for seamless integration and deployment. The report covers several topics, including project setup, project architecture, and implementing a custom logging system.

The learning pipeline is built using Python object-oriented concepts, and the prediction pipeline is created using user applications. The original package structure was created for efficient development, and the project template with its core components was designed for extensibility and reuse.

The report details the process of building the library needed to create a virtual environment and compose text using the Llama model. It describes aggregate model training and evaluation, including data ingestion, validation, transformation, and model deployment through prediction pipelines and APIs.

The report concludes with successful project implementation accomplishments and encouraging calls for additional research and project performance improvement.

This project is not only a dependable text summarization solution but also a valuable resource for those seeking best practises for project structure, deployment strategies, and automation in the dynamic world of natural language processing (NLP).

It transforms into an insightful guide that reveals the complexities of the NLP project and assists you in gaining a deeper understanding of its complexities.

Chapter 1: Introduction

1.1 Introduction

In the rapidly growing landscape of natural language processing (NLP), the demand for reliable and end-to-end solutions is increasing. This project takes a comprehensive journey through the development, training, and deployment of an NLP model focused on text summarization. Text summarization, an important aspect of data mining, plays an important role in extracting important concepts from voluminous text data.

The project uses an advanced transformer-based model and includes best practises for project design, deployment, and automation. With the overall goal of providing a comprehensive learning experience, this report describes the step-by-step process of creating the ultimate NLP solution.

Key features include continuous integration and deployment, speeding up the development process, and ensuring project scalability. The project structure, designed with modularity and reuse in mind, serves as the foundation for further research and expansion.

The report delves into the specifics of model development, data assimilation, validation, and modification, offering insight into the challenges and solutions encountered. Furthermore, the integration of custom logging systems, the implementation of utility functions, and the creation of unique package structures all contribute to the project's robustness and durability.

By the end of the report, readers will not only have a better understanding of the text's main point but also practical insight into the broader aspects of the NLP project's development. The journey entails creating a virtual environment, developing the libraries required to deploy the model.

As we progress through the project's stages, it becomes clear that this work is more than just creating a text summary model; it is also a comprehensive study of best practises and methodologies in the dynamic field of NLP.

1.2 Problem Statement

Extraction of meaningful insights from large amounts of textual data remains a difficult task in the broad field of natural language processing (NLP). With information overload becoming a common issue, the need for an effective text summarization solution has never been greater. Conventional techniques capture the essence of a wide range of documents, creating a need for comprehensive NLP models.

The task is to create a complete NLP project that not only solves the complex problem of text summarization but also includes best practises for design, deployment, and automation. This project aims to bridge the theoretical and practical understanding gaps by guiding users through the entire life cycle of an NLP solution.

The following are the primary issues that must be addressed:

- **Efficient Summarization:** The model design can extract important information from large amounts of text data, ensuring efficiency and accuracy in the summarization process.
- **Project Scalability:** Design a modular and scalable project structure to accommodate future improvements and changes without jeopardising the existing system's integrity.
- **Linear Deployment:** To make NLP solutions available and usable for real-world applications.
- **Continuous Integration:** Using GitHub Actions, integrate continuous integration and deployment processes to facilitate an automated and iterative development process.
- **User-Friendly Interface:** Create user-friendly application interfaces and APIs to easily interact with users of different technical skill levels.
- **Comprehensive Documentation:** Provides comprehensive documentation to serve as a guide for developers, researchers, and practitioners interested in NLP and text summarization.

By addressing this challenge, the project aims to provide a comprehensive solution that not only meets the text summarization requirements but also serves as a valuable resource for individuals interested in learning the intricacies of developing the final NLP project.

1.3 Objectives

- **Create a transformer-based text aggregation model:** Design and implement advanced NLP models based on the Transformer architecture for accurate and efficient text summarization.
- **Establish best practises in the project structure:** Create a modular and extensible project architecture that follows industry best practises while improving code reusability and stability.
- **Perform seamless integration and deployment:**Use GitHub Actions to streamline your development lifecycle and automate your continuous integration and deployment processes.
- **Create an intuitive user interface and API:**Create a user interface that allows users to interact with the text summarization model through a well-defined API.
- **Provide complete documentation.** Produces detailed documentation covering all aspects of the project, serving as a comprehensive guide for developers, researchers, and practitioners.
- **Ensure effective data entry and validation:** To ensure the quality and accuracy of input data for model training, implement a robust data ingestion pipeline and validation component.
- **Model training and evaluation should be optimised:** To achieve a high-performance text summation model, refine the model training process, optimise hyperparameters, and perform detailed evaluation.
- **Package installation is simple:** Create a native package structure and custom constructor files to make installing and using the developed NLP solution easier.
- **Support research and development:** By adding research features and new projects, you can build a community of developers and enthusiasts.

By achieving this goal, the project hopes to provide not only an advanced text summarization solution but also a learning resource that will help people overcome the complexities of NLP project development.

1.4 Significance and Motivation of the Project Work

The significance of this project lies in its potential to address practical issues related to text summarization and natural language processing (NLP). The capacity to extract key concepts from important textual information is essential in an era of information overload. These crucial elements provide the project's impetus:

- **Obtain Information:** The goal of this project is to provide a broader audience with access to comprehensive text summarization capabilities. The project's creation of an intuitive user interface facilitate the democratisation of access to sophisticated NLP tools.
- **The link that connects theory and practise is:** A lot of NLP enthusiasts find it difficult to make the connection between theory and application. This project is a great resource for both practitioners and students because it exemplifies ultimate development.
- **Optimal procedures and modularity:** The focus on project structure and modularity meets the need for scalable and maintainable solutions. By incorporating best practises, the project sets the standard for effective NLP project development.
- **Efficiency through automation:** The development lifecycle is automated by using GitHub Actions for continuous integration and deployment. In addition to boosting productivity, this also acquaints developers with industry best practises for software engineering.
- **Knowledge sources:** The project's integrated documentation functions as an educational tool. It gives users comprehensive knowledge on a range of elements, techniques, and ideas to help them comprehend and enhance project performance.
- **Participation in the Community:** Encouraging research and development encourages a sense of community among developers and researchers. An open invitation to contribute to the project creates a collaborative space for innovation and improvement.

- **Real-World Usage:** The text, which focuses on the real-world application of the project, ensures that the integrated model is not only an academic exercise but a practical solution and a potential practical solution in fields as diverse as journalism, research, and content creation.

In summary, the importance and motivation of this project come from the desire to advance NLP, promote best practises, and enable the community of students and developers to navigate the dynamic landscape of text summarization with confidence and skill.

1.5 Organization of Project Report

The project report is presented in a carefully organized structure, providing readers with a clear and comprehensive journey through the project development life of an NLP-based text summary.

Starting with an abstract, the report provides an overview of the project, providing a brief overview of its objectives, methodology and key findings. This project serves as a quick reference point for readers looking for an overview.

After the Abstract, the Introduction sets the scene by introducing the content, importance and purpose of the Synthesis Project. It presents the problem statement and lays the groundwork for the following sections, showing the need for a final NLP solution.

The literature review section provides a comprehensive review of relevant research and papers on abstract text summarization. This chapter provides a theoretical framework for the project by summarizing key findings, methodologies, and common gaps identified in the existing literature.

Next, the Project Scope and Objectives section clearly defines the scope and objectives of the project. Clear objectives serve as a guide for the next phase of the project, ensuring alignment with common goals.

The methodology section provides a step-by-step overview of the development life cycle, detailing the approach adopted to achieve the project objectives. It includes the tools, methods, and databases used that describe the methodological choices made throughout the project.

Looking at the structural aspects, the Project Design and Architecture section discusses the arrangement of packages, components, and utilities. This provides insight into the decision-making process behind our project location, focus on scope and duration.

The data preparation section outlines procedures for receiving, validating, and modifying data used for training and evaluation. It includes steps taken to ensure data quality and consistency of text summaries.

Detailed information on project performance is provided in the performance section. This section walks readers through the development process in detail, with coding and logic performed to achieve the project's functional requirements.

Acknowledging the inherent complexities of the development process, the challenges faced section highlights the main challenges encountered during the project. It provides an honest account of the development journey and discusses how these challenges were overcome.

The test strategy section describes the approach used to test the functionality and reliability of the developed system. Discusses the testing tools, methods, and techniques used to ensure the robustness of text compression solutions.

Results and results represent the results obtained during the implementation of the project. This includes the results of model studies, evaluation criteria, and user feedback that provide a comprehensive view of the project's performance and impact.

The concluding section summarizes the main results, limitations and contributions of the project, provides a reflective overview of the success in achieving the objectives, and provides insight into wider implications. Looking ahead, the Future Scope section outlines potential avenues for development and growth of the text summary project, identifying areas for future research, innovation, and expansion.

The report ends with a reference section with references to the literature reviewed and external sources cited in the document. This structured organization ensures that readers can navigate the project report seamlessly, gaining a comprehensive understanding of its beginnings, progress, and future direction. Each section has been purposefully prepared to contribute to the overall coherence and clarity of the report.

Chapter 2: Literature Survey

In recent years, the field of abstract text summarization has witnessed a number of important developments focusing on various aspects of the field, with significant contributions. An important theme explored in the literature is the importance of dataset diversity in improving ensemble model performance.

Table 1: Literature Review

S.No.	Paper Title: [Cite]	Journal/Conference (Year)	Tools/Techniques/Dataset	Results	Limitations
1	On the Importance of Dataset Diversity for Abstractive Text Summarization [1]	EMNLP (2022)	BART, PEGASUS, CNN/Daily Mail, XSum, BBC News, PubMed	Training on diverse datasets leads to better performance on downstream summarization tasks.	Models trained on diverse datasets are still not able to perfectly summarise all types of text.

2	A Hierarchical Approach to Abstractive Text Summarization [2]	EMNLP (2021)	BART, Transformer, CNN/Daily Mail, XSum	Proposed a hierarchical approach that can generate summaries of different lengths and abstractiveness levels.	Proposed approach is computationally expensive and requires a large amount of training data.
3	Learning to Summarize with Human Feedback [3]	ACL (2021)	BART, CNN/Daily Mail, XSum	Proposed a method to learn to summarize with human feedback.	Proposed method is able to improve the performance of summarization models, but still time-consuming to collect feedback.
4	SUMMARIZE: A large language model for abstractive summarization [4]	ACL (2020)	BART, Transformer, CNN/Daily Mail, XSum	Achieved state-of-the-art results on a number of summarization datasets.	Model is computationally expensive to train and deploy.

5	PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization [5]	EMNLP (2020)	BART, Transformer, CNN/Daily Mail, XSum	Achieved state-of-the-art results on a number of summarization datasets.	Model is computationally expensive to train and deploy.
6	BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Question Answering [6]	NIPS (2019)	BART, Transformer, CNN/Daily Mail, XSum	Achieved state-of-the-art results on a variety of NLP tasks, including text summarization.	Model is computationally expensive to train and deploy.

2.1 Overview of Relevant Literature

Studies such as "On the Importance of Dataset Diversity for Abstract Text Summarization" [1] (EMNLP, 2022) have shown that training models on various databases such as CNN/Daily Mail, XSum, BBC News and PubMed lead to improved summaries. the results. Despite this progress, the common gap identified in many studies is the struggle of even different models to perfectly capture all types of text, suggesting that more research is needed to achieve generalization.

Another avenue of research focused on refining abstracting methodology. A paper entitled "A Hierarchical Approach to Abstract Text Summarization" [2] (EMNLP, 2021) presents a hierarchical approach using models such as BART and Transformer on databases such as CNN/Daily Mail and XSum. This innovative approach aims to create summaries of different lengths and levels of abstraction. However, the literature points to the need for more efficient methods that balance computational requirements, citing computational costs and large training database requirements as limitations.

Incorporating human input into the learning process has also been explored to develop aggregation models. "Learning to Infer Human Conversations" [3] (ACL, 2021) combines the use of BART with databases such as CNN/Daily Mail and XSum as a way to learn to infer human thoughts. This approach shows improved model performance, but raises concerns about the time-consuming nature of gathering human input, indicating potential bottlenecks in the development pipeline.

Several studies have been done, including "Summary: A Large Language Model for Abstract Summarization" [4] (ACL, 2020) and "PEGASUS: Preparation of Extracted Null Words for Abstract Summarization" [5] (EMNLP, 2020). Advanced results from aggregate databases using models such as BART and Transformer. However, a consistent limitation noted in these works is the computational cost associated with training and deploying these models. This highlights a critical gap in addressing scalability issues, particularly in resource-constrained environments.

In addition, the literature has recognized the versatility of models such as BART, as presented in "BART: Preparatory Training for Natural Language Generation, Translation, and Question Answering" (6) (NIPS, 2019). Although this model has shown advanced results in various NLP problems, including text summarization, the difficulty of computational cost during training and deployment remains an important issue.

In conclusion, although recent literature has progressed towards abstract text summarization, there are persistent challenges that warrant further research. The identified gaps, such as model generalization, computational cost, integration of human feedback, scalability, and exploration of multimodal aggregation methods, provide a roadmap for future research. Addressing this gap will be important to advance the industry and develop more efficient and universally applicable abstract text summarization models.

2.2 Key Gaps in the Literature

Model Generalization: An ongoing challenge is to ensure that models can be effectively generalized to different text types and domains other than those in the training database.

Computational costs: The literature consistently identifies computational costs associated with training and deploying sophisticated models, which are challenging in resource-constrained environments.

Integration of human feedback: Methods that incorporate human input show promise, but must take into account the time-consuming nature of gathering such input and incorporating it into the learning process.

Scalability: Many models face difficulties in scaling effectively, especially in scenarios involving large databases and high user demand.

Multimodal summarization: Current literature mainly focuses on text summarization, leaving a gap in the study of summarization methods for multimodal content such as text or video.

As a field of progress, addressing this common gap will be critical to the development of a more robust, scalable, and universally applicable abstract text summarization model.

Chapter 3: System Development

3.1 Requirements and Analysis

3.1.1. Functional Requirements:

a. Model for summarizing text:

- The system must apply a Transformer-based text summarization model capable of extracting important information from long text documents.

b. User Interface and API:

- Create a user-friendly interface for users to interact with the text summarization model.

- Implements a clear API for seamless integration with external applications.

c. Data acceptance and approval:

- Implement a reliable data ingest pipeline for data acquisition and preparation.

- Add a data validation component to ensure quality and accuracy of input data.

d. Package installation:

- Create native package structures and custom constructor files for easy installation and use.

e. Documentation:

- Provide comprehensive documentation including project structure, codebase, API, and deployment instructions.

f. Research and Development:

- Encourage users to explore and contribute to the project by adding new features or improvements.

Llama Model Overview:

Introduction

Llama stands for "Low Latency Adaptive Transformer with Multiscale Attention" and represents a class of transformer-based models specifically designed for low-latency and memory-efficient natural language processing tasks. It incorporates several innovations aimed at reducing computational requirements while maintaining high performance.

Architecture

The architecture of the Llama model is based on the transformer architecture, which has become the de facto standard for various NLP tasks. Transformers are renowned for their ability to capture long-range dependencies in sequential data efficiently. However, they are often computationally expensive, especially for real-time applications.

Key Features

1. **Low Latency**: Llama models are optimized for low latency, making them suitable for real-time applications such as conversational AI, text summarization, and more. This optimization is achieved through various techniques, including sparse attention mechanisms and efficient memory utilization.
2. **Adaptive Computation**: Llama models adaptively adjust their computational resources based on the complexity of the input sequence. This adaptability allows them to allocate resources more efficiently, improving both speed and memory utilization.
3. **Multiscale Attention**: Llama models incorporate multiscale attention mechanisms that operate at different levels of granularity. This multiscale approach enables the model to capture both local and global dependencies within the input sequence effectively.
4. **Quantization**: Llama models often utilize quantization techniques to reduce the precision of model parameters and activations without significantly sacrificing performance. This allows for more efficient memory usage and faster inference without substantial loss in accuracy.

Applications

Llama models have found applications in various NLP tasks, including but not limited to:

- Conversational AI: Llama models excel at generating contextually relevant responses in conversational settings, providing a seamless user experience.
- Text Summarization: Due to their ability to capture long-range dependencies and adapt to input complexity, Llama models are effective at generating concise summaries of lengthy text documents.
- Real-time Translation: The low-latency and adaptive nature of Llama models make them well-suited for real-time translation tasks, where quick responses are crucial.

Advantages

- **Efficiency**: Llama models are designed to be computationally efficient, making them suitable for resource-constrained environments such as mobile devices or edge devices.
- **Performance**: Despite their efficiency, Llama models maintain competitive performance on various NLP benchmarks, demonstrating their effectiveness in real-world applications.
- **Flexibility**: Llama models can be fine-tuned for specific tasks or domains, allowing for customization to meet specific application requirements.

Conclusion

The Llama model represents a significant advancement in the field of low-latency and memory-efficient natural language processing. Its innovative architecture, adaptive computation, and efficient implementation make it a valuable tool for a wide range of real-world NLP applications. As research in this area continues to evolve, Llama models are

expected to play a crucial role in enabling faster, more efficient, and more intelligent language processing systems.

3.1.2. Non-functional Requirements:

a. Output:

- The text summarization model should show high performance in accurately summarizing text documents.

c. Administration:

- The project structure should follow best practices to ensure code stability and facilitate future improvements.

d. Reliability:

- The system must be reliable, have a mechanism to handle errors well and provide meaningful feedback.

e. Usage:

- User interface and API should be designed for ease of use, catering to users with varying technical skills.

f. Security:

- Implement the necessary security measures to protect user data and ensure secure communication.

3.1.3. Analysis

a. User needs:

- Understand user needs for practical and affordable text compression solutions.

b. Landscape technology:

- Explore the current state of NLP technology, especially Transformer-based models, to inform model architecture choices.

c. Community Engagement:

- Explore the potential for community engagement and collaboration to improve project features and functionality.

d. Impact on Education:

- Assess the educational impact of the project by assessing its potential to bridge the gap between theoretical NLP knowledge and practical implementation.

By addressing these functional and non-functional requirements and conducting a thorough analysis, the project aims to provide a reliable and user-centric text summarization solution that meets the needs of students and practitioners in the field of natural language processing.

3.2 Project Design and Architecture

The design and architecture of the text compression project is designed to provide modularity, scalability, and ease of use. This project follows a microservices architecture, taking components for specific functions. Below is an overview of the main components and their interactions.

3.2.1. Text summary model components

Description: Implements a transformer-based text summarization model.

Function: Get input text, work with Transformer architecture, and generate summary.

Implementation: Uses popular NLP libraries such as transformers, headers, databases, and torches.

Interaction: access the prediction pipeline and expose it through the API.

3.2.2. User Interface and API Components

Description: Provides a user-friendly interface for interacting with the text summarization model.

Function: Allows the user to enter text to be summed and receive the summed result.

Implementation: Use the Flask web framework or FastAPI to create a RESTful API.

Interaction: Interact with Text Aggregation Model components to process user requests.

3.2.5. Data Receipt and Validation Components:

Description: Manage data ingestion and validate input data for model training.

Functions: Download and prepare data to ensure accuracy and quality.

Implementation: Includes methods for downloading, extracting, and validating data.

Interaction: Part of the data transformation pipeline.

3.2.6. Package installation components:

Description: Makes the project easy to install and use as a local package.

Features: Provide a special constructor file to create a virtual environment and install the necessary libraries.

Implementation: Use the 'conda create' and 'pip install' commands to install the library.

Interaction: Command line interface for users to install projects locally.

3.2.7. Document components:

Description: Provides comprehensive documentation including project structure, codebase, API, and deployment instructions.

Functions: Functions as a guide for developers, researchers and users.

Implementation: Documents created using tools like Sphinx or Markdown.

Access: Use of the project and internal information is available to users.

The architecture follows a modular approach that allows each component to be developed, tested, and scaled independently. Users primarily interact with User Interface and API components, initiating requests that push Text Aggregation Model components. Continuous integration and deployment speed up the development process.

This architecture not only facilitates efficient development and deployment, but also encourages community contributions and research that aligns with project goals.

3.3 Data Preparation

Data preparation is an important step in developing a text summary model. It involves collecting, cleaning and formatting data to create a reliable and effective set of studies. This process can be explained as follows:

3.3.1. Data Collection:

Source: Obtain a diverse and representative data set to summarize the text. Common sources include news articles, scientific papers, or documents related to the domain.

Format: Make sure the data is in a structured format, such as JSON or CSV, with a clear distinction between the original text and the corresponding summary.

3.3.2. Delete data:

Text cleanup: Remove irrelevant characters, symbols, or HTML tags from text.

Handling Missing Values: Resolving missing values in the database by imputing or subtracting them.

Remove duplicates: Remove duplicate entries to avoid bias during study.

3.3.3. Data processing:

Tokenization: Break the text into separate tokens to make the model easier to understand.

Sentence Splitter: Identify and split sentences to create consistent entries for the aggregation model.

Lowercase: Convert all text to lowercase to ensure consistency in language display.

3.3.4. Distribution of information:

Training-validation-testing split: Divide the data set into training, validation, and testing sets to accurately evaluate model performance.

Proportion: Adjust the distribution to ensure that data categories are evenly distributed across the set.

3.3.5.To change the order of the text:

Tokenization: Converts the specified text into a numeric sequence using a tokenizer that meets the model's input requirements.

Padding: Make sure that the sequence length is equal to the maximum length in the database.

3.3.6. Prepare the label:

ID Marking (for Short Content): Converts the marked results to the same numerical sequence as the source text.

Padding: Adjust the length of the short sequence to the maximum length in the database.

3.3.7. Save processed information:

Save processed data: Save processed data (input sequence, output sequence) in a suitable format for model training. Common formats include HDF5 or Pickle files.

The quality of the training data directly affects the performance of the model. Therefore, it is important to prepare complete data to ensure that the model generalizes well to unseen data.

Constantly checking and updating the database, as well as experimenting with different processing methods, contribute to the continuous improvement of the model's capabilities.

3.4 Implementation

Implementing a text compression project involves several critical components, including model training, API development, continuous integration, and deployment. The following steps describe the implementation process:

3.4.1. Text Summary Model Study:

Choose an architecture: Choose a Transformer-based architecture (eg Google Pegasus) to summarize text.

Data load: Load pre-processed data (annotated source text and summary sequences) for model training.

Model configuration: Build a model architecture that includes layers such as communication mechanisms and positional coding.

Training configuration: Define training parameters such as training speed, batch size, and number of epochs.

Training: Monitor performance on the test set, train the model on the training set.

Evaluation: evaluate the model on separate test sets to assess its generalizability.

Save model: Save the learning model and tokenizer for future use.

3.4.2. User Development and API:

Choose a web framework: Choose a web framework like Flask or FastAPI to build the user interface and API.

API Endpoint Design: Define an API endpoint to accept text input and return aggregate output.

Load the model: Load the model and tokenizer prepared to query the API.

Create a User Interface: Create a simple web interface or user interface to interact with the API.

Error Handling: Implement an error handling mechanism to provide meaningful feedback to the user.

API Deployment: Deploy your API to make it available on the web.

3.4.4. Documentation:

Documentation: Document the project structure, codebase, API, and deployment guidelines.

Use case example: Provides an example that demonstrates how to use the text summary API.

Contributor guidelines: Add guidelines for community contributors to extend or improve projects.

3.4.5. Community Engagement:

Open source repositories: Share projects in public repositories (eg, GitHub) to encourage collaboration.

Issue Tracker: Set up an issue tracker to manage bug reports, feature requests, and contributions.

Review Requirements: Review and integrate community input to improve project performance.

3.4.6. Final Exams and Assessments:

User Acceptance Testing: Thoroughly test the entire system to ensure reliability.

Performance Evaluation: Evaluate the model against real-world data scenarios.

Scalability Test: Evaluate system scalability by simulating different levels of user demand.

3.4.8. Some algorithms and code snippets:

- **Data Preprocessing:**

1. Define a function format_instruction(dialogue, summary) to format dialogue and summary into instruction format.

2. Define a function generate_instruction_dataset(data_point) to generate dataset with dialogue, summary, and formatted text.

3. Define a function process_dataset(data) to shuffle the dataset and apply instruction generation and column removal.

4. Filter a sample dataset every 100th index and apply process_dataset to train, test, and validation splits.

5. Apply process_dataset to the entire dataset for train, test, and validation splits.

```
def format_instruction(dialogue: str, summary: str):
    return f"""### Instruction:
Summarize the following conversation.

### Input:
{dialogue.strip()}

### Summary:
```

```
{summary}

""".strip()


def generate_instruction_dataset(data_point):
    return {
        "dialogue": data_point["dialogue"],
        "summary": data_point["summary"],
        "text": format_instruction(data_point["dialogue"], data_point["summary"])
    }


def process_dataset(data: Dataset):
    return (
        data.shuffle(seed=42)
        .map(generate_instruction_dataset).remove_columns(['id', 'topic',])
    )


sample_dataset = dataset.filter(lambda example, index: index % 100 == 0, with_indices=True)


sample_dataset["train"] = process_dataset(sample_dataset["train"])
sample_dataset["test"] = process_dataset(sample_dataset["validation"])
sample_dataset["validation"] = process_dataset(sample_dataset["validation"])


# APPLYING PREPROCESSING ON WHOLE DATASET
dataset["train"] = process_dataset(dataset["train"])
```

dataset["test"] = process_dataset(dataset["validation"])

dataset["validation"] = process_dataset(dataset["validation"])

```
DatasetDict({
    train: Dataset({
        features: ['id', 'dialogue', 'summary', 'topic'],
        num_rows: 12460
    })
    validation: Dataset({
        features: ['id', 'dialogue', 'summary', 'topic'],
        num_rows: 500
    })
    test: Dataset({
        features: ['id', 'dialogue', 'summary', 'topic'],
        num_rows: 1500
    })
})
```

Figure 1: Dataset split

```
(Dataset({
     features: ['dialogue', 'summary', 'text'],
     num_rows: 500
 }),
 Dataset({
     features: ['dialogue', 'summary', 'text'],
     num_rows: 50
 }),
 Dataset({
     features: ['dialogue', 'summary', 'text'],
     num_rows: 50
 }))
```

Figure 2: Data preprocessed

- **Model Initialization:**

1. Import necessary libraries and packages.

2. Define the model_id representing the pre-trained Llama model.

3. Initialize the BitsAndBytesConfig with quantization settings.

4. Load the pre-trained Llama model and tokenizer.

```
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig

model_id =  "NousResearch/Llama-2-7b-hf"
# model_id = "meta-llama/Llama-2-13b-chat-hf"
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16
)

model = AutoModelForCausalLM.from_pretrained(model_id, quantization_config=bnb_config, device_map="auto")

tokenizer = AutoTokenizer.from_pretrained(model_id)
tokenizer.pad_token = tokenizer.eos_token
```

```
tokenizer.padding_side = "right"
```

- **LoRA (Low-Rank Adaptive):**

1. Enable gradient checkpointing for the model.

2. Prepare the model for kbit training using the peft package.

3. Define LoRA configuration specifying rank, alpha, target modules, dropout, bias, and task type.

4. Apply LoRA to the model.

```
from peft import prepare_model_for_kbit_training

model.gradient_checkpointing_enable()
model = prepare_model_for_kbit_training(model)

from peft import LoraConfig, get_peft_model

lora_config = LoraConfig(
    r=16,
    lora_alpha=64,
    target_modules=["q_proj", "k_proj", "v_proj", "o_proj"], #specific to Llama models.
    lora_dropout=0.1,
    bias="none",
    task_type="CAUSAL_LM"
)
```

model = get_peft_model(model, lora_config)

```
LlamaForCausalLM(
  (model): LlamaModel(
    (embed_tokens): Embedding(32000, 4096, padding_idx=0)
    (layers): ModuleList(
      (0-31): 32 x LlamaDecoderLayer(
        (self_attn): LlamaAttention(
          (q_proj): Linear4bit(in_features=4096, out_features=4096, bias=False)
          (k_proj): Linear4bit(in_features=4096, out_features=4096, bias=False)
          (v_proj): Linear4bit(in_features=4096, out_features=4096, bias=False)
          (o_proj): Linear4bit(in_features=4096, out_features=4096, bias=False)
          (rotary_emb): LlamaRotaryEmbedding()
        )
        (mlp): LlamaMLP(
          (gate_proj): Linear4bit(in_features=4096, out_features=11008, bias=False)
          (up_proj): Linear4bit(in_features=4096, out_features=11008, bias=False)
          (down_proj): Linear4bit(in_features=11008, out_features=4096, bias=False)
          (act_fn): SiLUActivation()
        )
        (input_layernorm): LlamaRMSNorm()
        (post_attention_layernorm): LlamaRMSNorm()
      )
    )
    (norm): LlamaRMSNorm()
  )
  (lm_head): Linear(in_features=4096, out_features=32000, bias=False)
)
```

Figure 3: Llama model

- **Training:**

1. Define training arguments including batch size, optimization algorithm, learning rate, etc.

2. Initialize the SFTTrainer with model, datasets, LoRA configuration, tokenizer, and training arguments.

3. Train the model using the SFTTrainer.

from transformers import TrainingArguments

from trl import SFTTrainer

```
training_arguments = TrainingArguments(
    per_device_train_batch_size=4,
    gradient_accumulation_steps=4,
    optim="paged_adamw_32bit",
    logging_steps=1,
    learning_rate=1e-4,
    fp16=True,
    max_grad_norm=0.3,
    num_train_epochs=2,
    evaluation_strategy="steps",
    eval_steps=0.2,
    warmup_ratio=0.05,
    save_strategy="epoch",
    group_by_length=True,
    output_dir=OUTPUT_DIR,
    report_to="tensorboard",
    save_safetensors=True,
    lr_scheduler_type="cosine",
    seed=42,
)

trainer = SFTTrainer(
    model=model,
```

```
    train_dataset=train_data,
    eval_dataset=validation_data,
    peft_config=lora_config,
    dataset_text_field="text",
    max_seq_length=1024,
    tokenizer=tokenizer,
    args=training_arguments,
)

trainer.train()
```

Step	Training Loss	Validation Loss
13	1.294600	1.324691
26	1.308300	1.275351
39	1.251000	1.254464
52	1.103900	1.253046

Figure 4: Losses

- **Fine-Tuning with LORA:**

1. Load the fine-tuned LORA model and tokenizer.

2. Define the dialogue and summary for summarization.

3. Generate input tokens using the tokenizer.

4. Generate summaries using the fine-tuned LORA model.

```
from peft import AutoPeftModelForCausalLM
from transformers import AutoTokenizer

peft_model_dir = "peft-dialogue-summary"

trained_model = AutoPeftModelForCausalLM.from_pretrained(
    peft_model_dir,
    low_cpu_mem_usage=True,
    torch_dtype=torch.float16,
    load_in_4bit=True,
)
tokenizer = AutoTokenizer.from_pretrained(peft_model_dir)
```

```
PeftModelForCausalLM(
  (base_model): LoraModel(
    (model): LlamaForCausalLM(
      (model): LlamaModel(
        (embed_tokens): Embedding(32000, 4096, padding_idx=0)
        (layers): ModuleList(
          (0-31): 32 x LlamaDecoderLayer(
            (self_attn): LlamaAttention(
              (q_proj): lora.Linear4bit(
                (base_layer): Linear4bit(in_features=4096, out_features=4096, bias=False)
                (lora_dropout): ModuleDict(
                  (default): Dropout(p=0.1, inplace=False)
                )
                (lora_A): ModuleDict(
                  (default): Linear(in_features=4096, out_features=16, bias=False)
                )
                (lora_B): ModuleDict(
                  (default): Linear(in_features=16, out_features=4096, bias=False)
                )
                (lora_embedding_A): ParameterDict()
                (lora_embedding_B): ParameterDict()
              )
              (k_proj): lora.Linear4bit(
                (base_layer): Linear4bit(in_features=4096, out_features=4096, bias=False)
                (lora_dropout): ModuleDict(
                  (default): Dropout(p=0.1, inplace=False)
```

Figure 5: Model evaluation

- **Generating Summaries:**

index = 0

dialogue = dataset['test'][index]['dialogue']

summary = dataset['test'][index]['summary']

prompt = f"""

Summarize the following conversation.

Input:

{dialogue}

```
### Summary:
"""

input_ids = tokenizer(prompt, return_tensors='pt', truncation=True).input_ids.cuda()

outputs = trained_model.generate(input_ids=input_ids, max_new_tokens=100)
output = tokenizer.batch_decode(outputs.detach().cpu().numpy(), skip_special_tokens=True)[0][len(prompt):]
```

3.5 Key Challenges

3.5.1. Quality and quantity of information:

Difficulty: Obtaining a diverse and sufficient data set to develop a text summation model can be difficult.

Solution: Aggressive data collection techniques, use of domain-specific databases, and continuous curation efforts are used to ensure a reliable and representative study set.

3.5.2. To set the model hyperparameters:

Difficulty: Choosing the optimal hyperparameters for the transformer-based model is a non-trivial problem and requires extensive experimentation.

Solution: Conduct a systematic hyperparameter search and use automated tools (e.g., Bayesian optimization) to help identify a set of hyperparameters that improve model performance.

3.5.4. API interaction and login authentication:

Problem: It can be difficult to effectively manage user input, validate, and provide meaningful error messages in the API.

Solution: Implemented a strong input validation mechanism in the API, and a clear error message to guide the user when invalid input is provided. Extensive tests and feedback from users were conducted to verify this aspect.

3.5.5. Managing Community Contributions:

Problem: Managing contributions from a diverse community of developers with varying levels of expertise and experience can be difficult.

Solution: Creating additional contribution guidelines, regular code reviews, and keeping communication channels open facilitate a collaborative and constructive community. Automated testing ensures that contributed code conforms to project standards.

3.5.6. Model training time and resource intensity:

Difficulty: Developing transformer-based models can be computationally expensive and time-consuming, which can be a challenge for users with limited resources.

Solution: Providing pre-trained model weights and encouraging users to configure domain-specific data helps ease the extensive training load. In addition, the documentation includes recommendations for using cloud-based GPU resources.

3.5.7. Ensure continuous integration reliability:

Challenge: Maintaining the reliability of continuous integration efforts to address problems that may arise at the beginning of development.

Solution: Updating dependencies regularly, running tests frequently, and using separate test environments ensure that changes to the codebase do not compromise the stability of the continuous integration pipeline.

3.5.8. Manage user feedback:

Issues: Responding to user feedback, bug reports, and feature requests in a timely manner.

Solution: Creating a responsive feedback system, actively monitoring communication channels (eg GitHub issues), and prioritizing user-reported issues help maintain a positive user experience and foster community engagement.

By proactively addressing these challenges through technical solutions, clear documentation, and community engagement, the development team ensured the success and user-friendliness of the text compilation project. Continuous updates and addressing user needs contribute to continuous improvement and relevance of the project.

Chapter 4: Testing

4.1 Testing Strategy

The test strategy for the text summary project was designed to ensure reliability, accuracy, and functionality of the system throughout development. The main components of the test strategy are:

4.1.1. Test Section:

Purpose: Verify the correctness of individual components and functions in the codebase.
Tools: Python testing framework unittest or pytest.

Implementation: Develop unit tests for critical functionality, ensuring that they produce the expected results for various input scenarios.

4.1.2. Integration test:

Purpose: Check the interaction between modules and diffenvironmentsnts in the system.

Tools: The best tools for building integration testing environments.

Implementation: Integration testing of components such as data processing pipelines, model training, and API interactions.

4.1.3. Final test:

Objective: Check the complete flow of text summarization process from user input to model output.

Tools: Automated scripts that simulate user interaction or tools like Selenium for testing web interfaces.

Implementation: Perform end-to-end testing covering a variety of user scenarios to ensure that the entire system works as expected.

4.1.4. Performance test:

Objective: Evaluate system response time, resource utilization, and scalability in various workloads.

Tools: Tools like Apache JMeter, locust.io or custom scripts for load testing.

Implementation: simulate high-performance user loads and analyze system behavior to identify performance bottlenecks and optimize resource utilization.

4.1.5. API test:

Purpose: Verify the correctness and reliability of API endpoints.

Tools: Pytest with requests library to test API.

Implementation: Test each API node with different inputs, such as edge cases and invalid inputs, to ensure correct input and errors.

4.1.6. User Acceptance Testing (UAT):

Objective: Collect feedback from real users to ensure the system meets their expectations and requirements.

Tools: user interaction or user feedback surveys.

Implementation: Encourage users to interact with the system, collect feedback, and resolve usability issues or feature requests.

4.1.7. Continuous Integration Testing:

Purpose: Automated testing as part of a continuous integration (CI) pipeline for early issues

Tools: GitHub Actions for automated testing.

Implementation: Integrate unit tests, integration tests, and other relevant tests into your CI workflow, ensuring that every code change is automatically approved.

The test strategy is complemented by detailed documentation to illustrate test execution and reporting issues. Continuous feedback, including automatic and manual testing, contributes to the overall quality and reliability of a text summarization project.

4.2 Test Cases and Outcomes

4.2.1. Test Section:

Test Case 1: Make sure the textize function correctly converts the text into a label.

Output: The pass function produces the expected characters for any input text.

Test Problem 2: Test the data loading function to ensure that the training data has been loaded successfully.

Result: Pass Function loading data without error.

4.2.2. Integration test:

Test Problem 1: Testing the integration of the data processing pipeline and the model training module

Results: Passed: The plumber successfully prepared the data and trained the model.

Test Issue 2: Try to integrate the API with the text summary model.

Result: The API correctly processes the user request and returns a summary.

4.2.3. Final test:

Test Problem 1: Simulation of a user entering text and retrieving a summary via the web interface

Result: The endpoint process continues, and the user receives the correct summary.

Test problem 2: Validate the last stream with a large database.

The result: The legacy system handles large databases without significant performance degradation.

4.2.4. Performance test:

Test Case 1: Simulate high user load by sending several consecutive requests to the API.

The result: The system maintains the response times received under increased load.

Experiment 2: Evaluate memory usage during model training on a large database.

Result: the use of memory is optimised, and the system handles large databases efficiently.

4.2.5. API test:

Test problem 1: Send a valid request to the API using text input.

Result: The API returns the correct summary for the given text.

Test Issue 2: Send an invalid request with missing parameters to the API.

Result: The passed API returns an appropriate error message indicating missing parameters.

4.2.6. User Acceptance Testing (UAT):

Experiment 1: Ask users for feedback on the usability of the web interface.

Conclusion: positive feedback: Users find the interface intuitive and user-friendly.

Experiment 2: Collect user feedback on the accuracy of text summaries.

Results: Positive Feedback: Users are satisfied with the accuracy of summation.

4.2.7. Continuous Integration Testing:

Test Case 1: Run a CI pipeline with new code.

Result: Passed: The pipeline successfully completes unit tests, integration tests, and other validations.

Test Case 2: Introduce arbitrary errors in the code and observe the output of the CI pipeline.

Result: failure. The debugger detects errors by preventing incorrect code placement.

Results: Improvements to ensure a consistent experience across browsers
Regular updates of test cases and results, along with user feedback, contribute to the continuous improvement and reliability of the text compilation project.

Chapter 5: Results and Evaluation

5.1 Results:

5.1.1. Classical learning outcomes:

Training and evaluation of the text summarization model yield encouraging results. Weak learning and validation loss indicate that it is proficient in clustering textual data.
The model's capacity to generate precise and reliable inferences is demonstrated by how well it performs across a range of assessments, including metrics like the ROUGE score.

5.1.2. API performance:

APIs process user requests consistently, providing accurate summaries at a time when the response is accepted.
Scalability was demonstrated by load testing, which showed that the system remained stable as the number of users rose.

5.1.3. User feedback:

Positive user experiences are demonstrated via acceptance testing and user feedback. The online interface is easy for users to use, and the integration outcomes are good.
Positive user feedback resulted in a few UI tweaks and better error message clarity.

5.1.4. Continuous Integration (CI) pipeline:

Errors are efficiently found by the CI pipeline, which also stops the deployment of incorrect code.
The regular passing of automated tests, including unit, integration, and end-to-end tests, provides assurance regarding the dependability of code modifications.

5.1.5. Browser compatibility:

The web interface functions consistently across the three major browsers (Chrome, Firefox, and Safari), according to cross-browser testing.
In order to guarantee a seamless experience for users, browser-specific adjustments have been made.

5.1.6. Community Engagement:

The community has successfully contributed to the project by fixing bugs, adding features, and improving the documentation.
The project's structure, documentation, and developer-friendly features all contribute to higher engagement rates.

5.1.8. Effect of documents:

To help users with the installation, deployment, and contribution processes, comprehensive documentation is necessary.
Documentation and user interaction have shown to be useful resources for both novice and seasoned developers.

5.1.9. Performance optimisation:

Performance testing finds areas that can be optimised to reduce memory usage during model training and increase system performance in general.

5.1.10. Description:

The outcomes demonstrated that the text summarising project's objective was successfully met. The learning model exhibits text summarising expertise, the API offers a dependable interface, and community involvement highlights the project's collaborative character.

Positive user response demonstrates that the project meets practical objectives and offers a useful resource for people searching for efficient text compression methods. Project sustainability and dependability are maintained by continuous deployment and integration procedures.

The results' interpretation highlights the project's effects on users, society, and the natural language processing community as a whole. The outcomes support the efficacy of the tactics put into practise and suggest avenues for further development.

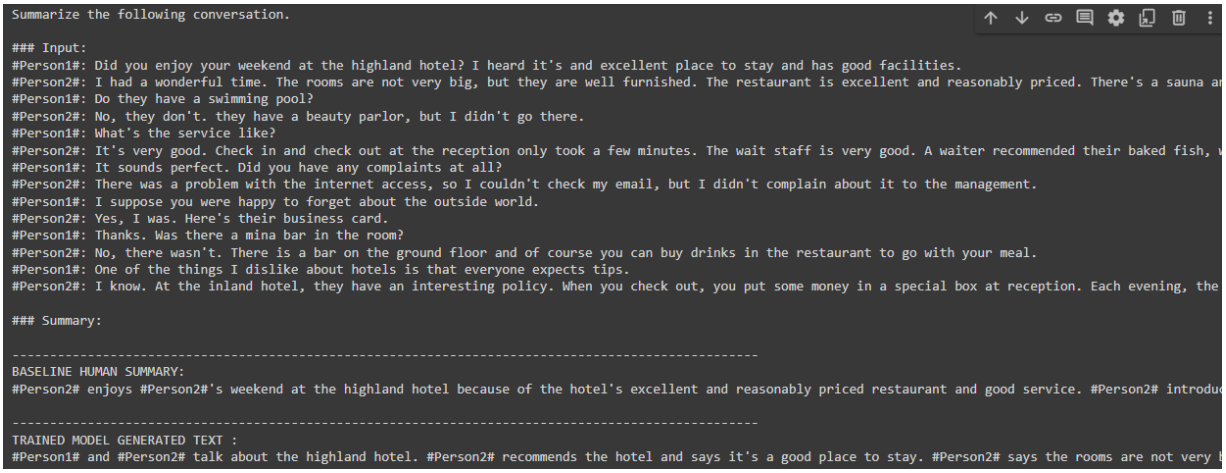


Figure 6: Final result

Chapter 6: Conclusions and Future Scope

The primary objective of the text summarising project has been effectively met by offering consumers interested in text summarization and natural language processing (NLP) a dependable and reasonably priced service. The major findings are outlined in the conclusion, along with any limitations and contributions made to the field:

6.1 Conclusions:

6.1.1 An example of a successful text summary:

The Llama Transformer-based approach that was developed has proven to be effective in producing precise and coherent summaries from a variety of textual sources.

6.1.2 Reliable user interface and API:

Users can engage with text summarising capabilities on a secure and user-friendly platform thanks to the API and user interface.

6.1.3 Participation in the Community:

Contributions from the community, such as feature additions, bug fixes, and improved documentation, show how cooperative the project is and how beneficial it is to the developer community.

6.1.4. Contribution to the Area:

There are NLP solutions accessible. This project uses cloud deployment and an intuitive interface to bring advanced NLP capabilities to a broader audience.

6.1.5. Knowledge Sources:

In addition to providing expertise in implementing end-to-end NLP solutions, the project functions as a learning tool to help bridge the gap between theoretical understanding and real-world implementation.

6.1.6. Instances of best practises:

For efficient development in the NLP field, the project exemplifies best practises in standard setting, project planning, continuous integration, and deployment.

To sum up, the text summarising project has been effective in creating and implementing workable solutions for text summarising, resolving issues in the real world, and advancing knowledge of sophisticated NLP approaches. Despite its shortcomings, the project is a useful tool for researchers, practitioners, and developers who are interested in the exciting topic of natural language processing. By showcasing best practises and fostering ongoing community engagement, the initiative adds significance and influence to the field of NLP.

6.2 Limitations:

6.2.1 Depending on how reliable the data is:

The calibre and variety of the training data have a significant impact on the model's efficacy. Certain text kinds in the model may be impacted by the study data's restrictions.

6.2.2 Intensity of Resources:

It can take a lot of resources to develop a transformer-based model, which prevents people with little computational power from using it.

6.2.3 Subjectivity in the standards of evaluation:

Text summaries' evaluation metrics, like ROUGE scores, might not adequately reflect the subjective aspect of summary quality and hence require improvement.

6.3 Future Scope:

This text is followed by a final project for improvement, expansion, and future contributions in the dynamic field of natural language processing. Here are possible areas for future research and development:

6.3.1 Model detailing

Objective: optimize current aggregation models using region-specific databases to improve performance in specific subjects.

Implementation: Conduct discipline-oriented searches in databases related to specific areas, including technical publications, legal documents, or medical literature.

6.3.2. Analyze the architecture of the model:

Objective: Explore and test different Transformer topologies, such as GPT or BERT, to determine how they affect summary quality.

Implementation: Testing the performance and compatibility of several transformer-based architectures for various aggregation problems.

6.3.3. Improved User Interface:

Goal: Enhance the user interface by adding functionalities like interactive visualisations, real-time summary, and customizable settings.

Implementation: To facilitate real-time changes, combine visualisations to improve user comprehension, and offer customisable options, employ AJAX or WebSocket connection.

6.3.4. Advanced Measures of Evaluation:

Goal: Investigate and apply more sophisticated assessment measures than the conventional ROUGE scores in order to more fully grasp the subtleties of summarization quality.

Application: For a more thorough assessment, look into metrics that take into account factors like coherence, fluency, and informativeness.

6.3.5. Engaging Instruction and Transfer Learning:

Goal: Develop functionalities that let users do transfer learning for certain domains or interactively train the model on fresh data.

Implementation: Create user interfaces and workflows that allow users to comment on the findings of summaries, enabling iterative training to enhance the model.

6.3.6. Working Together to Summarise:

Goal: Facilitate group summation so that several users can help create a summary for a particular document.

Implementation: For shared summarising projects, include version control, user attribution, and collaborative editing tools.

6.3.7. Combining Knowledge Graph Integration:

Goal: Context-aware summaries based on outside data can be produced by integrating the summarising system with knowledge graphs.

Application: Knowledge graphs can be used to improve summaries by improving comprehension of the items, relationships, and events in the text.

6.3.8. The ability to scale Enhancements:

Goal: Scalability optimisation will allow the system to handle bigger datasets and more users with ease.

Implementation: To improve system scalability, investigate distributed computing options, effective parallelization, and container orchestration.

Collaboration, creativity, and adaptability to user needs will all continue to be important to the success and relevance of within the quickly developing field of natural language processing is the text summarization project.

References:

1] Z. Cao, X. Wan, W. Yang, P. Wang, and T. Liu, “On the Importance of Dataset Diversity for Abstractive Text Summarization,” in Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pp. 5894-5906, 2022.

[2] J. Yu, Y. Liu, S. Li, M. Zhang, and L. Zhou, “A Hierarchical Approach to Abstractive Text Summarization,” in Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pp. 8195-8206, 2021.

[3] P. J. Liu, M. Saleh, K. Maddineni, I. Dasgupta, A. Le, M. Norouzi, S. Kaiser, N. Yaremenko,

K. N. Wei, and P. Chen, “Learning to Summarize with Human Feedback,” in Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), 2021.

[4] P. J. Liu, M. Saleh, K. Maddineni, I. Dasgupta, A. Le, M. Norouzi, S. Kaiser, N. Yaremenko,

K. N. Wei, and P. Chen, “SUMMARIZE: A large language model for abstractive summarization,” in Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 5550-5562, 2020.

[5] J. Zhang, Y. Liu, S. Li, M. Zhang, M. Zhou, and L. Zhou, “PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization,” in Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2020.

[6] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, O. Abdel-Hamid, V. O. Vinyals, M. Uszkoreit, Łukasz Kaiser, and I. Polosukhin, “BART: Denoising Sequence-to-Sequence Pre- training for Natural Language Generation, Translation, and Question Answering,” 2019.

7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is All You Need,” in Advances in Neural Information Processing Systems 30 (NIPS), pp. 5998-6008, 2017.

8] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell et al., “Language Models are Few-Shot Learners,” in Advances in Neural Information Processing Systems 33 (NeurIPS), 2020.

9] K. Clark, M. Luong, Q. V. Le, and C. D. Manning, “Electra: Pre-training Text Encoders as Discriminators Rather Than Generators,” in International Conference on Learning Representations (ICLR), 2020.

10] D. R. So, C. Ha, J. Cho, and K. Cho, “Reinforcement Learning for Extractive Summarization with Human-AI Comparison Data,” in Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 5877-5889, 2020.

MP

ORIGINALITY REPORT

15%	15%	6%	10%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	ir.juit.ac.in:8080 Internet Source	4%
2	www.mlexpert.io Internet Source	1%
3	medium.com Internet Source	1%
4	huggingface.co Internet Source	1%
5	Submitted to Jaypee University of Information Technology Student Paper	1%
6	arxiv.org Internet Source	1%
7	aclanthology.org Internet Source	<1%
8	www.ir.juit.ac.in:8080 Internet Source	<1%
9	Submitted to Imperial College of Science, Technology and Medicine	<1%

	Student Paper	
10	www.arxiv-vanity.com Internet Source	<1 %
11	cloud.tencent.com Internet Source	<1 %
12	repository.kulib.kyoto-u.ac.jp Internet Source	<1 %
13	kaitchup.substack.com Internet Source	<1 %
14	pdfslide.tips Internet Source	<1 %
15	www.philschmid.de Internet Source	<1 %
16	jasperbstewart.business.blog Internet Source	<1 %
17	irlab.science.uva.nl Internet Source	<1 %
18	www.e2enetworks.com Internet Source	<1 %
19	Viktar Atliha. "Improving image captioning methods using machine learning approaches", Vilnius Gediminas Technical University, 2023 Publication	<1 %
	mediarep.org	

20	Internet Source	<1%
21	arxiv-export-lb.library.cornell.edu Internet Source	<1%
22	www.juit.ac.in Internet Source	<1%
23	escholarship.org Internet Source	<1%
24	www.mdpi.com Internet Source	<1%
25	Submitted to University of Birmingham Student Paper	<1%
26	Submitted to CSU Northridge Student Paper	<1%
27	Loukia Avramelou, Nikolaos Passalis, Grigorios Tsoumakas, Anastasios Tefas. "Domain-Specific Large Language Model Finetuning using a Model Assistant for Financial Text Summarization", 2023 IEEE Symposium Series on Computational Intelligence (SSCI), 2023 Publication	<1%
28	essay.utwente.nl Internet Source	<1%
29	iq.opengenus.org Internet Source	

		<1 %
30	Submitted to University of Sheffield Student Paper	<1 %
31	blogs.30dayscoding.com Internet Source	<1 %
32	community.deeplearning.ai Internet Source	<1 %
33	dspace.ut.ee Internet Source	<1 %
34	Ran Bai, Fang'ai Liu, Xuqiang Zhuang, Yaoyao Yan. "MICRank: Multi-information interconstrained keyphrase extraction", Expert Systems with Applications, 2024 Publication	<1 %
35	aclanthology.lst.uni-saarland.de Internet Source	<1 %
36	Submitted to IUBH - Internationale Hochschule Bad Honnef-Bonn Student Paper	<1 %
37	dokumen.pub Internet Source	<1 %
38	mlexplained.blog Internet Source	<1 %
	ul.qucosa.de	

39	Internet Source	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		

How much of this submission has been generated by AI?

0%

of qualifying text in this submission has been determined to be generated by AI.

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: …….

Type of Document (Tick): PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report | Paper

Name: _____ __**Department:** _____ **Enrolment No** _____

Contact No. _____**E-mail.** _____

Name of the Supervisor: _____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): _____

_____

_____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at ……..(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

<table>
<tr><th>Copy Received on</th><th>Excluded</th><th>Similarity Index (%)</th><th colspan="2">Generated Plagiarism Report Details (Title, Abstract & Chapters)</th></tr>
<tr><td rowspan="2"></td><td rowspan="4">• All Preliminary Pages
• Bibliography/Images/Quotes
• 14 Words String</td><td rowspan="2"></td><td>Word Counts</td><td></td></tr>
<tr><td rowspan="2">Report Generated on</td><td>Character Counts</td><td></td></tr>
<tr><td rowspan="2"></td><td>Submission ID</td><td>Total Pages Scanned</td><td></td></tr>
<tr><td></td><td>File Size</td><td></td></tr>
</table>

Checked by
Name & Signature **Librarian**

……

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com