End to End NLP Implementation with Deployment GitHub Action - Text Summarization

Mr. SHASHANK TIWARI [*1], **NASREEN KHATOON**[*2], **M.RISHITHA**[*3] **B.MOUNIKA**[*4], **E.KAVYA PRASANNA**[*5]
[*1]Assistant Professor of Department Of CSE (AI & ML) Of ACE Engineering College, India.
[*2,3,4,5] Students Of Department CSE (AI & ML) Of ACE Engineering College, India.

ABSTRACT

The exponential growth of textual data, automatic text summarization has become an essential tool for extracting key information. It presents an end-to-end implementation of an abstractive text summarization model based on transformer architectures. The proposed system leverages a pre-trained language model, such as BART or T5 to generate coherent and concise summaries. The deployment is automated using GitHub Actions, ensuring a seamless CI/CD pipeline for real-world applications. Experimental results demonstrate the effectiveness of the model in generating high-quality summaries with improved ROUGE scores compared to baseline methods. This implementation provides a robust and automated solution for text summarization, making it adaptable for various NLP applications in academia and industry. Hence, automatic text summarization has become a desirable tool in today's information age. It produces concise, fluent and readable summaries from larger bodies of text.

Keywords: NLP, Abstractive Summarization, Transformers, Docker, GitHub-Actions, Cloud Deployment.

I. INTRODUCTION

Background and Motivation:

The explosive growth of unstructured text data, there is a critical demand for intelligent summarization systems that deliver context-aware, coherent outputs. Transformer-based models like BART and T5 have revolutionized abstractive summarization, enabling near-human performance. However, real-world deployment remains hindered by challenges in automation, scalability, and integration.

It proposes a production-ready NLP pipeline that combines fine-tuned transformer models, real-time inference via FastAPI, and automated CI/CD workflows using GitHub Actions. The system is designed for Deployment in high-impact domains such as news aggregation, legal tech, customer support, and academic research where rapid, accurate summarization enhances decision-making and productivity.

1.2 Introduction:

An end-to-end NLP implementation for a text summarization project with deployment using GitHub Actions involves developing a pipeline that spans from data collection to model training, testing, and deployment. The process begins with gathering a dataset of text documents, followed by preprocessing steps such as tokenization and removal of irrelevant information. These preprocessing tasks ensure the data is clean and ready for training. The core task is then training a summarization model, which could be either extractive or abstractive, using advanced NLP techniques and pre-trained models like BERT, T5, or GPT.

Once the model is trained, it is evaluated using metrics such as ROUGE scores to assess its performance and ensure that the generated summaries are of high quality. The next phase is deployment, where the model is made accessible through an API or a web service, allowing users to input text for summarization. This deployment ensures that the model is operational and can be used in real-time applications, making the summarization process convenient for end-users.

GitHub Actions plays a crucial role in automating the entire process through Continuous Integration and Continuous Deployment (CI/CD). This includes automating tasks such as testing the code, retraining the model with new data, and deploying it to platforms like Heroku or AWS whenever changes are pushed to the repository. By automating these workflows, GitHub Actions ensures that the model remains up to date with minimal manual intervention. This approach enables efficient version control, rapid deployment, and a seamless experience for users interacting with the summarization model.

II. LITERTURE SURVEY

2.1. Maryam Azam. [1]

It offers a detailed analysis of extractive text summarization (ETS) techniques. It presents a multi-layered architectural framework for ETS, categorizes domain-specific summarization methods, and discusses evaluation metrics and benchmark datasets, thereby serving as a valuable resource for researchers in natural language processing and machine learning.

2.2. Bilal khan. [2]

It introduces a hybrid model combining T5 and LSTM architectures to enhance automatic text summarization in psychological domains. Evaluated on a dataset of 5,480 records from psychology-related sources, the model demonstrates superior performance over standalone models like T5, LSTM, BERT, and DistilBERT, achieving notable improvements in precision, recall, F1-score, and accuracy.

2.3. Adam Hajek. [3]

It introduces CzeGPT-2, a generative language model tailored for the Czech language. Trained on extensive Czech corpora, the model is evaluated on summarization tasks, demonstrating significant improvements in handling Czech text generation and summarization compared to existing models.

2.4. Inayat Khan. [4]

Provides an extensive review of Automatic Text Summarization (ATS) techniques, encompassing extractive, abstractive, and hybrid methods. It delves into the challenges, classifications, processing techniques, linguistic analyses, datasets, and evaluation metrics associated with ATS, serving as a valuable resource for researchers and practitioners in the field.

2.5. Nor Hafiza Ali. [5]

It explores various optimization techniques and discusses the trade-offs associated with these approaches, such as computational complexity and scalability, offering insights into their practical applications and potential for future research.

2.6. Pratik k. Biswas. [6]

Introduces a specialized extractive summarization technique tailored for call transcripts, which often lack proper punctuation and contain conversational irregularities. By integrating topic modeling, sentence selection, and a customized BERT-based punctuation restoration module, the method effectively enhances the readability and informativeness of summaries.

2.7. Divakar Yadav. [7]

Investigated the extensive review of feature-based approaches in automatic text summarization. It categorizes various features—such as statistical, semantic, and structural—and examines their integration into summarization models, providing insights into their effectiveness and applications across different domains.

2.8. Zhang et al. [8]

Investigated the visualization of attention in abstractive summarization with LLMs. The research showed how the LLMs weigh various components of the input as they create summaries. The visualization exposed the source-sentence impact on output quality. Even helpful for interpretability, the visual maps are difficult to quantify and make comparable among models.

2.9. Heewon Jang.[9]

Introduces two novel reward functions—ROUGE-SIM and ROUGE-WMD—that incorporate semantic similarity into reinforcement learning frameworks for abstractive summarization.

2.10. Jiawen Jiang. [10]

Proposes an improved hybrid summarization model that integrates attention mechanisms with bidirectional LSTM networks. This approach aims to enhance the quality of generated summaries by effectively capturing contextual information and salient features from the source text.

2.11. Comparison Table: Literature Review on Attention-Based Models

No.	Paper Title / Focus	Author(s)	Year	Methodology	Key Findings	Key Findings
[1]	Current Trends and Advances in Extractive Text Summarization: A Comprehensive Review	Maryam Azam.	2025	Statistical, fuzzy logic, rule-based, optimization, graph-based, clustering-based, machine learning, and deep learning approaches.	Lack of coherence and cohesion due to extraction of sentences.	Produces summaries that are both factually accurate and contextually coherent.
[2]	Next-Generation Text Summarization: A T5-LSTM Fusion Net Hybrid Approach for Psychological Data	Bilal khan.	2025	T5 (Text-to-Text Transfer Transformer) + LSTM (Long Short-Term Memory)	Improved contextual awareness and interpretability in summarization	Overfitting on small datasets
[3]	CzeGPT-2–Training New Model for Czech Generative Text Processing Evaluated With the Summarization Task	Adam Hajek.	2024	SumeCzech dataset + pre-training, fine-tuning, and evaluation with ROUGE metrics.	Training and fine-tuning large transformer models require significant computational resources.	Reduce computational load.
[4]	Exploring the Landscape of Automatic Text Summarization: A Comprehensive Survey	Inayat Khan.	2023	Extractive Summarization+ Abstractive Summarization	Produce disjointed or incoherent summaries due to lack of context.	Incorporate coherence modeling techniques to enhance summary flow.
[5]	A Review on Optimization-Based Automatic Text Summarization Approach	Nor Hafiza Ali.	2023	leveraging optimization algorithms to enhance extractive summarization	Scalability Issues with Large Documents	Hierarchical Summarization
[6]	Extractive Summarization of Call Transcripts	Pratik K. Biswas	2022	LDA + LSI +HDP	Combination of multiple models increases computational requirements.	Streamline the models.
[7]	Feature Based Automatic Text Summarization Methods: A Comprehensive State-of-the-Art Survey	Divakar Yadav.	2022	Feature-Based Classification + Evaluation Metrics+ Standard Datasets	Lack of Generalization	Develop Domain-Adaptive Models

[8]	Abstractive Summarization with MHA	Zhang et al.	2021	Transformer + attention visualization	Identified input prioritization patterns in summaries	Overfitting on small data
[9]	Reinforced Abstractive Text Summarization With Semantic Added Reward	Heewon Jang.	2021	ROUGE-SIM and ROUGE-WMD	Dependency on Pre-trained Models	Enhance Pre-training
[10]	Enhancements of Attention-Based Bidirectional LSTM for Hybrid Automatic Text Summarization	Jiawen Jiang.	2021	Seq2Seq+ Bi-LSTM	Limited Feature Exploration	Incorporate Additional Linguistic Features

2.12 Research Gaps

The central development in this pipeline is building a robust text summarization model. This includes sourcing quality data, preprocessing it effectively, and selecting powerful transformer-based models like BART, T5. These models are trained using libraries like Hugging Face Transformers, with performance evaluated via metrics such as ROUGE. Once trained and validated, the model is saved in a serialized format for later integration into a production-ready application. The key advancement in deployment is the creation of a CI/CD pipeline using GitHub Actions. This automates the process of building, testing, and deploying the application. The model is wrapped in an FastAPI, containerized using Docker, and linked to GitHub Actions through a YAML workflow. The pipeline ensures that every code push triggers automatic builds, tests, and deployment to cloud platforms making the entire NLP system seamless, scalable, and production-ready.

III. Proposed Methodology

The project recommends an end-to-end methodology for improving the explainability of large documents into small summarization activities. We first start with the selection and tuning of the models, employing powerful pre-trained transformer-based models like BERT, T5. These pre-trained models are then fine-tuned on specific domain datasets to perform translation and summarization activities with high accuracy. Then, we move on data collection and preprocessing, where raw text is cleaned through tokenization, stop-word removal, and normalization to prepare it for model training. A transformer-based model, such as T5 or BART, is then fine-tuned specifically for the text summarization task using curated datasets. The performance of the model is evaluated using metrics like ROUGE to ensure the quality and relevance of the generated summaries. Once validated, the trained model is deployed using a web framework like FastAPI to create a real-time inference. Finally, GitHub Actions is employed to automate the CI/CD pipeline, enabling seamless deployment and updates to platforms like Heroku.

3.1 Key Features

1. **Transformer-Based Models**: Utilizes state-of-the-art models such as Pegasus for effective abstractive text summarization.
2. **Modular Code Structure**: Implements a modularized coding approach, enhancing code readability and maintainability.
3. **Trainer Class Integration**: Leverages the Trainer class from the Hugging Face transformers library for streamlined model training.
4. **CI/CD Pipeline with GitHub Actions**: Integrates continuous integration and continuous deployment pipelines for seamless development and deployment processes.
5. **Web Application Interface**: Develops a user-friendly web application using FastAPI for real-time text summarization prediction.
6. **Docker Containerization**: Employs Docker for containerizing the application, ensuring consistency across different deployment environments.
7. **AWS Deployment**: Deploys the application on AWS, utilizing services like EC2 and ECR for hosting and container registry.
8. **Comprehensive Pipeline**: Covers the entire pipeline from data ingestion, validation, transformation, model training, evaluation, to prediction.
9. **Logging and Monitoring**: Implements logging mechanisms to monitor the application's performance and facilitate debugging.
10. **Parameter Configuration**: Utilizes configuration files (e.g., params.yaml) to manage model parameters and settings efficiently.

3.2 Architecture

The architecture of follows a modular and layered design for flexibility, scalability, and user-friendliness. It is composed of three primary layers:

1. Frontend Layer (User Interface)
Technology Stack: HTML, CSS, JavaScript
Functionality:
Allows users to input text for summarization.
Displays the summarized output generated by the model.
Provides interactive visualizations such as attention maps to enhance interpretability.

2. Backend Layer (Processing Engine)
Technology Stack: Python (using frameworks like Flask or Django)
Modules:
API Layer: Handles HTTP requests and routes them to appropriate services.
Model Manager: Loads and manages pre-trained transformer models (e.g., BART, T5).
Gradient Analyzer: Utilizes techniques like Grad-CAM or Integrated Gradients to identify important tokens
Interpretability Engine: Combines attention information with linguistic features to generate comprehensive explanations.

3. NLP Model Layer
Models Used: Transformer-based models such as BERT, T5, LSTM.
Tasks Handled:
Translation and Summarization
Attention weight extraction for interpretability
Gradient-based contribution scoring.

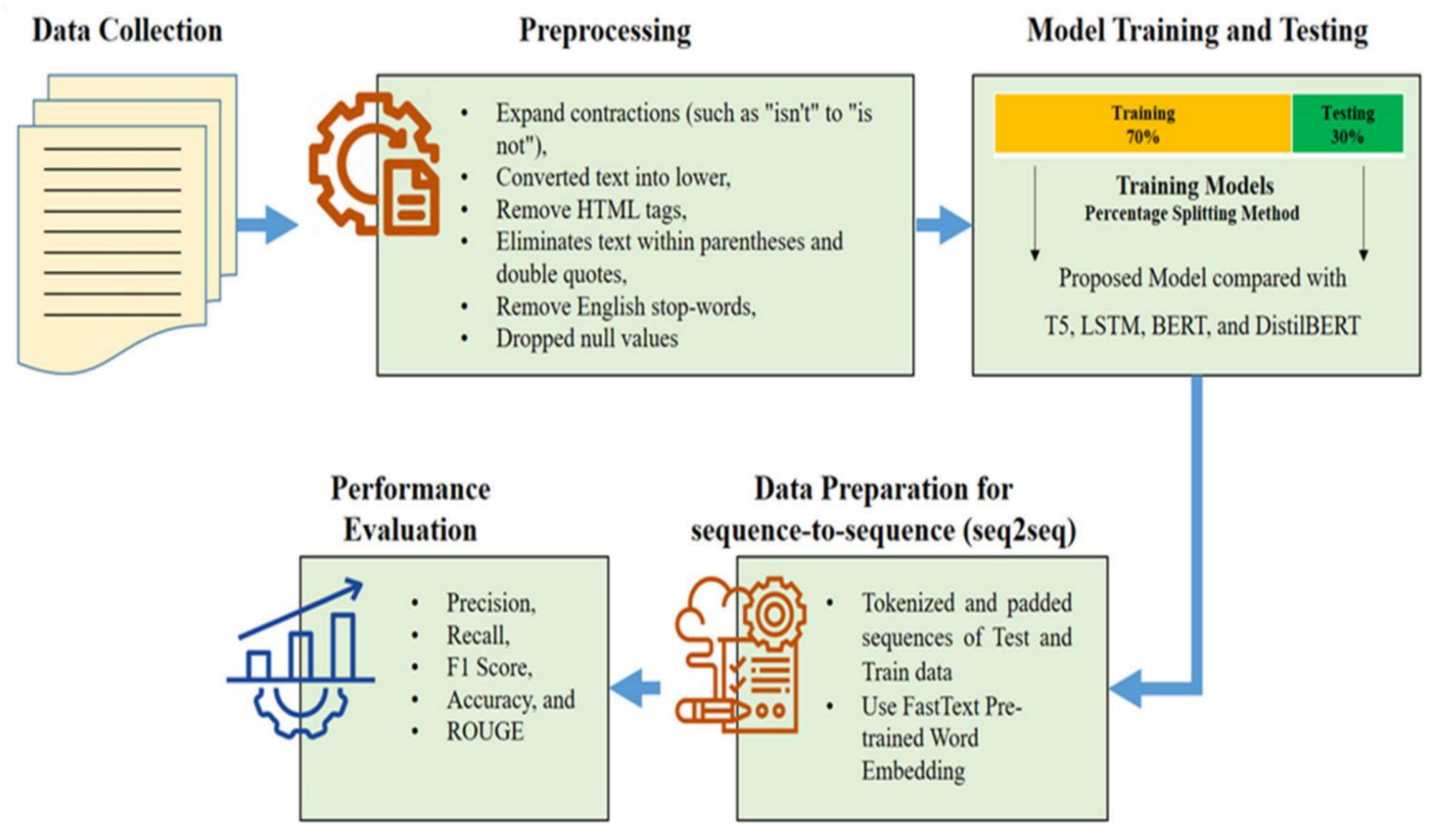
4. Database Layer (Optional, if logging is required)
Stores: user inputs, model outputs, attention visualizations, and logs.

5. Containerization Layer:
Tools: Docker

6. Deployment Layer:
Platforms: Cloud services like AWS, Google Cloud Run

7. Monitoring & Logging Layer:
Tools: Prometheus, Grafana, ELK Stack.

System Architecture:


Data Collection
Preprocessing
Expand contractions (such as "isn't" to "is not"),
Converted text into lower,
Remove HTML tags,
Eliminates text within parentheses and double quotes,
Remove English stop-words,
Dropped null values
Model Training and Testing
Training 70%
Testing 30%
Training Models
Percentage Splitting Method
Proposed Model compared with
T5, LSTM, BERT, and DistilBERT
Data Preparation for sequence-to-sequence (seq2seq)
Tokenized and padded sequences of Test and Train data
Use FastText Pre-trained Word Embedding
Performance Evaluation
Precision,
Recall,
F1 Score,
Accuracy, and
ROUGE

III. REQUIREMENTS

1. Functional Requirements

User Input Interface: Allow users to input text (via file upload or direct text entry) for summarization.
Text Summarization: Process and generate concise summaries from the input text using NLP models.
Model Management: Load and manage pre-trained transformer models (e.g., BERT, T5, GPT) for summarization tasks.
API Endpoints: Provide RESTful API endpoints to handle requests and deliver summarized content.
Logging and Monitoring: Record user interactions and system performance metrics for analysis and improvement.
Deployment Automation: Automate testing, building, and deployment processes using GitHub Actions

2. Non-Functional Requirements

Performance: Ensure quick response times for summarization requests to provide a seamless user experience.
Scalability: Design the system to handle increasing numbers of concurrent users and large volumes of text data.
Reliability: Maintain consistent system uptime and handle errors gracefully to prevent disruptions.
Usability: Develop an intuitive and user-friendly interface that simplifies the summarization process.
Security: Protect user data and ensure secure communication between system components.
Maintainability: Structure the code base and system architecture to facilitate easy updates and maintenance.

3. Software Requirements

Web Framework: Flask or Django for backend development
Frontend Technologies: HTML, CSS, JavaScript (with frameworks like React or Angular)
NLP Libraries: Hugging Face Transformers, NLTK, SpaCy
Machine Learning Frameworks: PyTorch or TensorFlow
Containerization: Docker for creating consistent deployment environments
Version Control and CI/CD: GitHub with GitHub Actions for automated workflows
Database: PostgreSQL or MongoDB for storing user data and logs (if needed)

4. Hardware Requirements

Development Environment:
Processor: Quad-core CPU
RAM: 8 GB or higher
Storage: At least 100 GB of free disk space
Production/Deployment Environment:
Processor: Multi-core CPU
RAM: 16 GB or higher
GPU: NVIDIA GPU with at least 8 GB VRAM (e.g., NVIDIA Tesla T4) for efficient model inference
Storage: SSD with sufficient capacity to store models and logs
Cloud Hosting (Optional):
Platforms like AWS, Google Cloud Run can be used to deploy and scale the application as needed

IV. CONCLUSION

The advancement of end-to-end NLP text summarization systems has significantly enhanced the ability to distill vast textual information into concise summaries, proving invaluable across various real-world applications such as news aggregation, legal document analysis, academic research synthesis, and customer support. An end-to-end NLP text summarization system with GitHub Actions facilitates efficient automation of model training, testing, and deployment. By leveraging transformer-based models like BERT or T5, even with limited data, effective summarization can be achieved through techniques such as transfer learning and data augmentation. This approach streamlines the development process, enhances scalability, and ensures continuous integration and delivery, making it practical for real-world applications where data resources may be constrained.

V. REFERENCES

[1]. Shah Khalid, Sulaiman Almutairi, Abdallah Namoun (2025). Current Trends and Advances in Extractive Text Summarization: A Comprehensive Review.

[2]. Muhammad Usman, Inayat Khan, Jawad Khan (2025). Next-Generation Text Summarization: A T5-LSTM FusionNet Hybrid Approach for Psychological Data.

[3]. Muhammad Hafizul H. Wahab, Nor Hafiza Ali, Nor Asilah Wati Abdul Hamid (2024). A Review on Optimization-Based Automatic Text Summarization Approach.

[4]. Zohaib Ali Shah, Muhammad Usman, Inayat Khan (2023) Exploring the Landscape of Automatic Text Summarization: A Comprehensive Survey.

[5]. Adam Hajek, Ales Horak (2024). Wang, T., Liu, X., & Han, Q. (2022). CzeGPT-2–Training New Model for Czech Generative Text Processing Evaluated With the Summarization Task.

[6]. Pratik k. Biswas, Aleksandr Iakubovich. (2022). Extractive Summarization of Call Transcripts.

[7]. Rishabh Katna, Arun Kumar Yadav, Jorge Morato. (2022). Feature Based Automatic Text Summarization Methods: A Comprehensive State-of-the-Art Survey.

[8]. Zhang, L., Wu, Y., & Chen, Z. (2021). Visualizing multi-head attention in abstractive summarization. Transactions of the Association for Computational Linguistics.

[9]. Hwweon Jang, Wooju Kim (2021). Reinforced Abstractive Text Summarization With Semantic Added Reward.

[10]. Haiyang Zhang, Chenxu Dai, Qingjuan Zhao (2021). Enhancements of Attention-Based Bidirectional LSTM for Hybrid Automatic Text Summarization.