*A mini project report on*

# GENERAL NEWSLETTER GENERATOR

*submitted in partial fulfillment of the requirement for the award of the degree of*

**BACHELOR OF ENGINEERING**

**in**

**COMPUTER SCIENCE & ENGINEERING**


Submitted By

Ms K.Mithila 245322733030

Mr M.Sri Lohith 245322733059


Under the Guidance of

**Mr. A. Vivek Reddy**
**Assistant Professor**


**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

## NEIL GOGTE INSTITUTE OF TECHNOLOGY

(Approved by AICTE, New Delhi & Affiliated to Osmania University, Hyderabad)
Kachavanisingaram Village, Hyderabad, Telangana 500058


August, 25

# NEIL GOGTE INSTITUTE OF TECHNOLOGY

A Unit of Keshav Memorial Technical Education (KMTES)
Approved by AICTE, New Delhi & Affiliated to Osmania University, Hyderabad

**Dated:**

## CERTIFICATE

This is to certify that the Mini Project Report titled **GENERAL NEWSLETTER GENERATOR** being submitted by **Ms K. Mithila ,Mr M. Sri Lohith** bearing HT. NO **245322733030 ,245322733059** in partial fulfillment for the award of Bachelor of Engineering in Computer Science & Engineering to the Osmania University is a record of bonafide mini project work carried out by him/ her under our guidance and supervision.

The results embodied in this project work have not been submitted to any other University or Institute for the award of any degree or diploma.

**Mr. A. Vivek Reddy**
Project Guide
Assistant Professor, Dept. of CSE
Neil Gogte Institute of Technology

**Dr. P. VAISHALI**
Head of the Department
Dept. of CSE
Neil Gogte Institute of Technology

**External Examiner**

# NEIL GOGTE INSTITUTE OF TECHNOLOGY

A Unit of Keshav Memorial Technical Education (KMTES)
Approved by AICTE, New Delhi & Affiliated to Osmania University,
Hyderabad

**Dated:**

## DECLARATION

I/We hereby declare that the Mini Project entitled, "**GENERAL NEWSLETTER GENERATOR**" has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The Mini project is done in partial fulfillment of the requirements for the award of degree of

**BACHELOR OF ENGINEERING (COMPUTER SCIENCE and ENGINEERING)** to

be submitted as sixth semester Mini project as part of our curriculum.

**Name and Signature of the Student**

**K. Mithila 245322733030**

**M. Sri Lohit 245322733059**

# ACKNOWLEDGEMENT

The successful completion of this project would not have been possible without the support and guidance of several individuals, to whom we are deeply grateful.

We would like to express our sincere gratitude to our guide, **Mr. A. Vivek Reddy Assistant Professor**, for their constant support, valuable feedback, and clear guidance throughout the development of this mini project. Their inputs helped us better understand the technical aspects and stay focused on our goals.

We are also thankful to our mini project coordinator **Dr. T. R. Srinivas,** for consistently assisting us and ensuring that the project requirements and timelines were well-managed. Their efforts made the entire process smooth and well-organized.

We extend our thanks to **Dr. P. Vaishali,** Head of the Department of Computer Science and Engineering, for providing us with the facilities, encouragement, and academic environment needed to carry out this work effectively.

Finally, we would like to thank our Principal, **Prof. R. Shyam Sunder,** for their support and for promoting a positive atmosphere that encourages innovation and hands-on learning among students.

# Table of contents

# List of Figures

# List of Tables

# ABSTRACT

In the digital age, people face a constant stream of news and information from many sources. This makes it hard to filter, understand, and keep up with what matters. Manually sorting through articles takes a lot of time and is not very effective, especially when users only want high-quality, brief insights from a flood of long articles. This project introduces a Daily Newsletter Generator. This automated system collects real-time news articles through RSS feeds and scrapes full content using Newspaper3k. The articles are then processed and summarized using DistilBART, a model designed for summarizing text. DistilBART is a smaller, faster version of Facebook's BART model.

The application uses Flask for backend and frontend integration. It allows users to input topics and get tailored, summarized news digests. This system reduces reading time and ensures accessibility and relevance by filtering articles based on user-selected keywords. The final output is a newsletter-style interface that gives quick and accurate news insights across various categories, such as politics, science, health, and technology etc.

By using web scraping, filtering text, and summarizing with deep learning, this project provides a practical solution for information overload. It also sets the stage for future improvements like personalized newsletters, support for multiple languages, and integration with scheduling or email services for daily delivery.

**Keywords:** News Summarization, Abstractive Summarization, DistilBART, Transformer Models, NLP, RSS Feeds, Flask, Real-time News, Deep Learning, Text Processing, Web Scraping, Newsletter Automation.

# CHAPTER 1: INTRODUCTION

## 1.1BACKGROUND AND CONTEXT

In today's digital world, the sheer amount of online news and information makes it hard for people to stay informed without spending a lot of time and effort. Thousands of articles come out each day on various topics. Users need smart systems that can quickly filter, summarize, and present relevant information in a clear and concise way.

NewsFlash tackles this issue by using advancements in Natural Language Processing (NLP), especially abstractive text summarization, to turn long news articles into brief, clear summaries. Unlike traditional extractive methods that pull together key sentences from the original text, abstractive models like DistilBART create human-like summaries that more effectively capture the main ideas and context of the article.

This system helps users engage with news more efficiently. It shows how transformer-based models can be used in real-time summarization tasks. By integrating RSS news scraping, smart topic filtering, and advanced summarization processes, NewsFlash offers a fast, scalable, and user-friendly solution for understanding current events. This is ideal for readers, researchers, and professionals who need to stay updated without feeling overwhelmed by information.

## 1.2PROBLEM STATEMENT

In today’s digital age, people are overwhelmed by news from many online sources. From breaking headlines to opinion pieces and in-depth reports, it’s become harder for readers to stay informed without spending a lot of time filtering, reading, and understanding lengthy content. Traditional newsletters, while convenient, often take a one-size-fits-all approach. They lack personalized topics, content summaries, and ways to engage users. They usually include unfiltered links or full articles, which puts the responsibility of reading and understanding entirely on the user. This leads to inefficient news consumption, and often, important stories being missed. Students, professionals, and everyday readers need a system that helps them stay updated without sacrificing time or mental energy.

The NewsFlash project tackles this issue by creating a personalized newsletter system that pulls in the latest news articles in real-time from RSS feeds based on topics chosen by users. It uses natural language processing (NLP) techniques and transformer-based models like DistilBART to produce clear, concise summaries from lengthy articles. This way, users get the main points of each story

without having to read the whole piece. The platform displays these summaries in a simple and user-friendly web interface, allowing users to quickly scan through multiple articles. Additionally, the system can send these summaries as a newsletter via email or as downloadable files, giving users flexibility in how they receive the news. Users can even select their preferred level of summarization, ranging from brief abstracts to summaries similar to full articles. Overall, NewsFlash makes news consumption smarter, faster, and more personalized, transforming overwhelming news feeds into an efficient and engaging experience.

## 1.3OBJECTIVES OF THE PROJECT

- To develop and design a real-time topic-based content aggregation system.
- To deploy an abstractive summarization system based on the DISTILBART transformer model.
- To incorporate web scraping methods for scraping relevant article content.
- To implement a responsive web interface for user input and generation of newsletters.
- To test the efficiency of the system through functional and user testing.

## 1.4 SCOPE OF THE PROJECT

The General Newsletter Generator collects and summarizes news articles based on topics chosen by users. It pulls real-time content using RSS feeds and retrieves full articles with the Newspaper3k library. The system filters and cleans the raw text by removing HTML tags and unnecessary data. It uses DistilBART, a lightweight model for summarizing content. This process gives users short, context-rich summaries instead of full articles. The results appear on a clean, responsive web interface created with Flask and styled with Tailwind CSS. Users no longer need to sort through long articles. The initial system focuses on gathering, summarizing, and presenting content based on topics. Features like support for multiple languages or fake news detection are not included in the current version. Overall, the project provides a personalized and efficient news experience.

## 1.5 METHODOLOGY

The General Newsletter Generator project has a modular and scalable structure that makes news aggregation, processing, and presentation easier. The system uses an iterative method which allows for ongoing testing, evaluation, and improvement of each module. News articles are collected in real time through RSS feeds and scraped with the Newspaper3k library. The raw HTML content is then cleaned and prepped to remove any irrelevant data.

The cleaned text goes to the DistilBART (sshleifer/distilbart-cnn-12-6) model from Hugging Face for summarization. This model creates short, human-like summaries. The whole backend, which includes scraping, text cleaning, and summarization, is built in Python using Flask as the web framework. The application can run locally or be temporarily deployed on platforms like Google Colab with ngrok tunneling for public access.

The frontend is built with HTML and CSS, using Tailwind CSS for optional styling. It communicates with the backend through Flask-rendered templates. Users input topics using a web form. The backend fetches, summarizes, and displays news articles specific to those topics. The modular design makes it possible to add features in the future, such as PDF summarization, support for multiple languages, and email newsletters.

## 1.6 SIGNIFICANCE OF THE PROJECT

The General Newsletter Generator tackles the problem of too much information by providing a practical and automated way to summarize news in real time. In a time when readers are often overwhelmed by excessive online content, this project offers a quick, dependable, and personalized method to stay informed. It is especially important for students, professionals, and researchers who need timely insights in specific areas without dedicating hours to reading entire articles.

## 1.7 ASSUMPTIONS, CONSTRAINTS, AND TARGET AUDIENCE

### 1.7.1 Assumptions

The system assumes that all news articles retrieved through RSS feeds are in English and have grammatically correct content suitable for summarization. It assumes that the sources used, like Google News, provide accessible and free URLs that can be processed with the Newspaper3k library. The project also relies on the availability and stability of third-party APIs and models, such as DistilBART from Hugging Face, and assumes these services stay accessible during runtime.

It is also assumed that users provide valid and relevant topic inputs and do not try to submit harmful or incorrectly formatted queries. For future additions, such as PDF uploads, the system will assume that uploaded documents are not encrypted, password-protected, or image-based scanned files.

### 1.7.2 Constraints

This application is constrained based on computational resources, particularly when running on platforms like Google Colab that enforce session timeouts and memory restrictions. It only supports English text and cannot handle scanned PDFs, non-selectable text, or offline use. Model inference might be slow depending on the input size and GPU availability. The quality of summaries relies on how clear the input is. There is no grammar correction or error handling for low-quality content.

### 1.7.3 Target Audience

The main users of the system are students, teachers, and professionals who need quick summaries of news and information. It works well for workers, researchers, and content creators who want to stay updated on certain topics without having to read full-length articles. The tool also benefits non-technical users because its web-based interface is simple, responsive, and easy to use, making it available to a wide range of audiences, no matter their technical skills.

# CHAPTER 2: SYSTEM ANALYSIS

## 2.1 Existing System

Most news aggregation and summarization systems use extractive summarization techniques like TF-IDF, TextRank, or frequency-based sentence scoring. These methods choose and compile key sentences directly from the source. While they are quick and cheap to run, they often create summaries that are disjointed, repetitive, and lacking in context. They do not rephrase content or ensure coherence, which lowers the overall readability and usefulness of the summaries.

Furthermore, many traditional systems do not support real-time news retrieval or topic-specific filtering. They often rely on pre-defined categories or fixed sources, which lack personalization and flexibility. Tools that provide summarization in email newsletters usually do this with generic headlines or short metadata, rather than using deep learning-based summaries. As a result, users still have to read the full articles for a better understanding, which defeats the purpose of saving time.

## 2.2 Proposed System

The proposed system, NewsFlash, is an AI-powered news summarization platform that provides concise, topic-based news digests using real-time RSS feeds. It integrates DistilBART (sshleifer/distilbart-cnn-12-6), a transformer-based model optimized for summarizing articles, to create clear summaries from full-length news articles**.**

Users enter their topics of interest through an easy web interface. The backend, built with Flask, retrieves relevant articles using Feedparser and scrapes the full content with Newspaper3k. The extracted text is cleaned and sent to the summarization model, which provides short, clear summaries.

The output appears in a clean, responsive interface styled with HTML and CSS. It provides an easy experience for users. The application runs locally or on platforms like Google Colab, with optional ngrok tunneling for public access. NewsFlash shortens reading time, keeps the focus on topics, and gives a flexible way to stay informed without overwhelming users with information.

## LLM MECHANISM

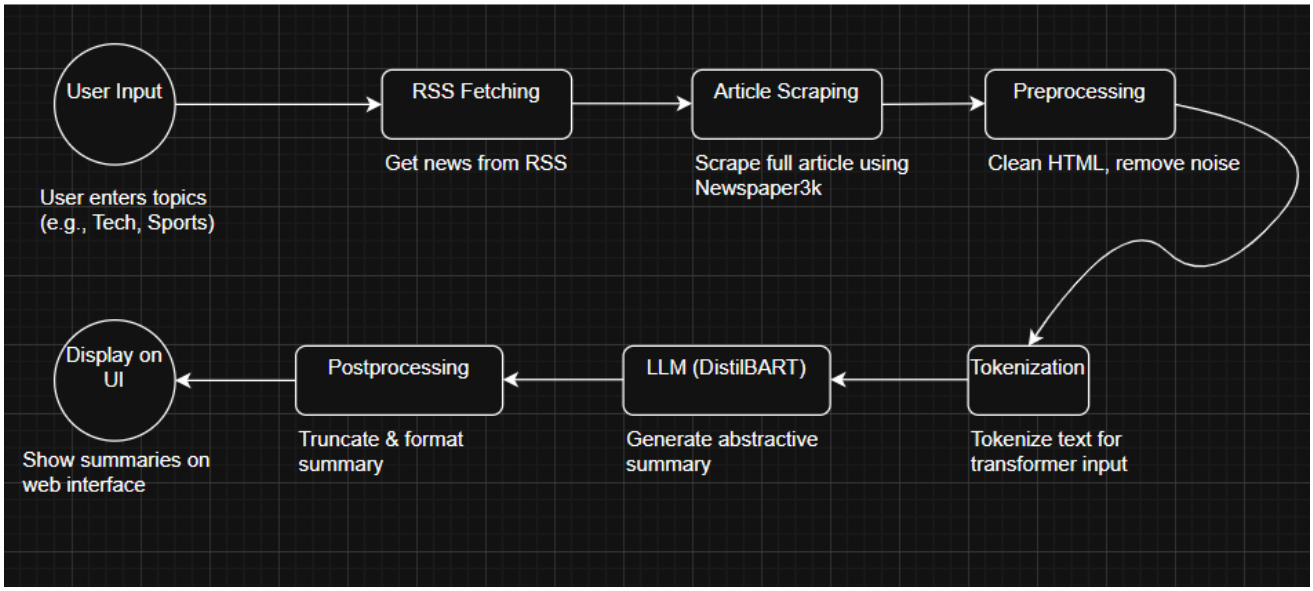



Fig 2.1 LLM MECHANISM

This diagram illustrates the LLM-based summarization pipeline in the NewsFlash project. Users input topics, which trigger RSS fetching and article scraping using Newspaper3k. The scraped content is preprocessed to remove noise, then tokenized for transformer compatibility. The DistilBART model generates abstractive summaries from tokenized input. These summaries are postprocessed and displayed on the web interface for the user.

## ARCHITECTURE OF BART MODEL

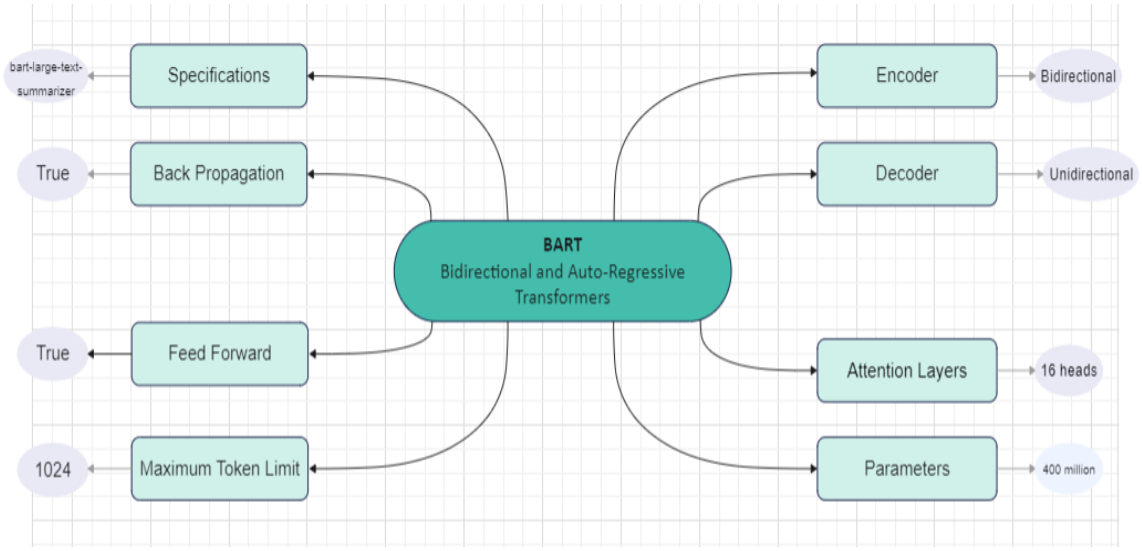



Fig 2.2 ARCHITECTURE OF BART MODEL

The figure presents the encoder–decoder architecture of the BART model, which is used for abstractive summarization. It explains how the model is pre-trained by corrupting the input and learning to reconstruct it. The flow of input through the encoder and the generation of output from the decoder is shown clearly. This structure enables BART to generate coherent and fluent summaries. It is one of the core models used in the backend of our system.

### 2.2.1 Key features:

- **Modular Python backend** using Flask for routing, article processing, and summarization.
- **Web-based user interface** built with HTML/CSS and integrated via Flask templates (no React/REST used).
- **Real-time news scraping** from RSS feeds and full-article extraction using Newspaper3k.
- **Abstractive summarization** using the DistilBART model from Hugging Face Transformers.

The system offers an intuitive interface, personalized topic input, and quick access to summarized news. It is lightweight, interpretable, and suitable for users in education, media monitoring, and professional domains.

## 2.3 Feasibility Study

### 2.3.1 Technical Feasibility

- The project uses the DistilBART model from Hugging Face. It is pre-trained and lightweight, allowing it to run on free Google Colab environments with GPU support.
- The Flask backend enables efficient routing, integrates summarization models, and generates responses quickly.
- The frontend is built with HTML and CSS, rendered through Flask templates. This makes it simple, responsive, and easy to deploy.
- Tools like Newspaper3k and Feedparser handle article scraping and RSS parsing reliably.

### 2.3.2 Economic Feasibility

- The system uses free and open-source tools like Google Colab, Hugging Face Transformers, Feedparser, and Newspaper3k to keep costs low.
- Ngrok allows for temporary public hosting without the need to buy a domain or server.
- There is no dependency on paid APIs or proprietary software. This makes the solution affordable and easy to access for students and small teams.
- The infrastructure needs are minimal, requiring only basic hardware and internet access for development and deployment.

### 2.3.3 Operational Feasibility

## 2.4 Project Specifications

- The system has a simple, user-friendly web interface that lets even non-technical users create news summaries easily.
- Although it is mainly designed for news articles, it can also manage other text inputs in future updates.
- The solution fits well into workflows in education, research, and business, helping users stay informed efficiently.
- The system’s design allows for future growth, such as email delivery, personalization, or mobile access.

### 2.4.1 Input Types

- Plain text input through a web form where users enter topic keywords, such as “technology, sports, health.”

### 2.4.2 Output Types

- Abstractive summaries of news articles are generated using the DistilBART model. These summaries are written in a natural, human-like style.

### 2.4.3 Backend Logic

- Article fetching → Text extraction → Cleaning → Model inference (DistilBART) → Output formatting for display.
- It uses Flask routing to handle user topic input and manage the summarization workflow.
- Separate functions handle RSS feed parsing, web scraping (via Newspaper3k), and summarization.
- Summarized results are formatted and displayed through the Flask.

#### 2.4.4 Frontend Logic

- Users can enter topic keywords using a simple, responsive HTML form.
- The interface shows real-time summarized news articles which include titles, summaries, and article links.
- It communicates with the Flask backend through form submission.
- The frontend is created with HTML and CSS, designed for readability and accessibility on all devices.

## 2.5 Project Requirements

### 2.5.1 Hardware Requirements

Table 2.1 Hardware Requirements

| Component | Specification |
|---|---|
| CPU | Intel Core i5 or higher |
| RAM | Minimum 8 GB |

| | |
|---|---|
| GPU | NVIDIA T4 (via Google Colab for DistilBART summarization) |
| Storage | 100 GB SSD (for temporary files and scraped article content) |
| Network | Stable high-speed internet (for RSS fetching, Colab access, and ngrok tunneling) |

### 2.5.2 Software Requirements

Table 2.2 Software Requirements

| **Category** | **Tools/Libraries used** |
|---|---|
| Operating System | Windows 10/11 or Linux (for local development) |
| Frontend | HTML, CSS |
| Backend | Python 3.8+, Flask (web framework) |
| Model Libraries | transformers, torch, sentencepiece (for DistilBART) |
| News Parsing | feedparser, newspaper3k, lxml |
| API & Server | Flask, pyngrok (for public tunneling on Colab) |
| Cloud Runtime | Google Colab (for hosting backend and models) |

# CHAPTER 3: LITERATURE SURVEY

**1. Liu and Lapata (2019) – "Text Summarization with Pretrained Encoders" (Published by EMNLP IJCNLP, Volume 1, pp. 3728–3738)**

Existing Work:

This work introduced BERTSUM, a model that builds on the BERT encoder for extractive summarization by adding sentence-level classification layers. The model finds important sentences in the document and combines them to create summaries. It did well on benchmark datasets like CNN/DailyMail and The New York Times.

The frontend uses HTML and CSS, designed for readability and accessibility across devices.

Limitations of Existing Work:

As an extractive method, BERTSUM directly pulls sentences from the source. This results in fragmented summaries that have limited fluency and no paraphrasing. It does not support abstractive capabilities, which are important for natural, human-like summaries.

**2. Nallapati et al. (2016) – "Abstractive Text Summarization using Sequence to Sequence RNNs and Beyond" (Published by ACL, CoNLL, pp. 280–290**

Existing Work:

This paper proposed an abstractive summarization model using encoder-decoder RNNs. It included pointer-generator and coverage mechanisms. These features helped manage out-of-vocabulary words and cut down on repetition. The model performed better than many extractive systems of that period.

Limitations of Existing Work:

The model, which is based on RNNs, struggled to capture long-term dependencies. It also had slower training and inference times. Factual consistency was an issue in longer documents.

**3. See et al. (2017) – "Get To The Point: Summarization with Pointer Generator Networks" (Published by ACL, Volume 1: Long Papers, pp. 1073–1083)**

Existing Work:

This paper presented a hybrid model that combines sequence-to-sequence generation with a pointer-generator method and coverage loss. The model can copy from the source while also generating new words. This capability results in more fluent summaries.

Limitations of Existing Work:

Despite improvements, it still used RNNs. This limited training speed and scalability when compared to modern transformer-based models.

**4. Vaswani et al. (2017) – "Attention Is All You Need" (Published by NeurIPS Volume 3: pp.234-340)**

Existing Work:

This landmark paper introduced the **Transformer** architecture, which relies solely on self-attention mechanisms and positional encoding, eliminating the need for recurrence. It laid the foundation for highly parallelizable and scalable models like BERT, GPT, BART, and T5.

Limitations of Existing Work:

The initial Transformer model was powerful, but it focused on machine translation. It required a lot of computing power, which made it less suitable for low-resource settings without fine-tuning.

**5. Lewis et al. (2020) – "BART: Denoising Sequence-to-Sequence Pre training" (Published by Facebook AI pp.129-378)**

Existing Work:

BART combines BERT's encoder and GPT's decoder in one structure. It is pretrained using a denoising autoencoder objective. Fine-tuned BART has performed very well on text generation tasks like summarization and question-answering.

Limitations of Existing Work:

BART has a fixed token-length limit, necessitating chunking for long inputs. It also requires fine tuning for specific domains to maintain performance quality.

**6. Raffel et al. (2020) – "Exploring the Limits of Transfer Learning with a Unified Text to Text Transformer" (Published by Google Research pp.23-145)**

Existing Work:

Proposed T5, a model framing all NLP tasks as text to text transformations. T5 achieved state of the art results across tasks like summarization, translation, and QA. The authors released multiple model sizes and conducted detailed ablation studies on pretraining strategies.

Limitations of Existing Work:

Its performance relies heavily on prompt design and large-scale computing resources. In low-resource settings or without careful tuning, the model's efficiency and output quality can drop.

**7. Du and Cardie (2017) – "Learning to Ask: Neural Question Generation for Reading Comprehension" (Published by ACL Volume 5 pp.479-567)**

Existing Work:

This early neural model for question generation used answer-aware RNNs to encode context and

generate grammatically correct and relevant questions, outperforming rule-based systems.

Limitations of Existing Work:

The use of RNNs made it less efficient and scalable. It also lacked adaptability across domains and struggled with longer or more complex inputs.

**8. Zhou et al. (2018) – "Neural Question Generation from Text: A Review" (Published by Springer pp.324-576)**

Existing Work:

This comprehensive review surveyed the evolution of question generation models, discussing rule-based, statistical, and neural approaches. It provided insights into datasets, model limitations, and evaluation metrics.

Limitations of Existing Work:

As a survey paper, it did not propose new methodologies. It noted that many existing models lacked generalization capability, were highly dataset-dependent, and produced low-diversity outputs.

# CHAPTER 4: SOFTWARE DESIGN

Software design is the process of defining the architecture, components, modules, and interfaces needed to meet the system's functional and non-functional requirements. For this project, which focuses on real-time abstractive news summarization, the design emphasizes modularity, efficiency, and seamless integration across the frontend, backend, and model inference layers. Each layer is set up to ensure scalability, easy maintenance, and a smooth user experience.

## 4.1 System Architecture

The system architecture is organized into three primary layers: the frontend (client-side), the backend (server-side), and the AI model layer.

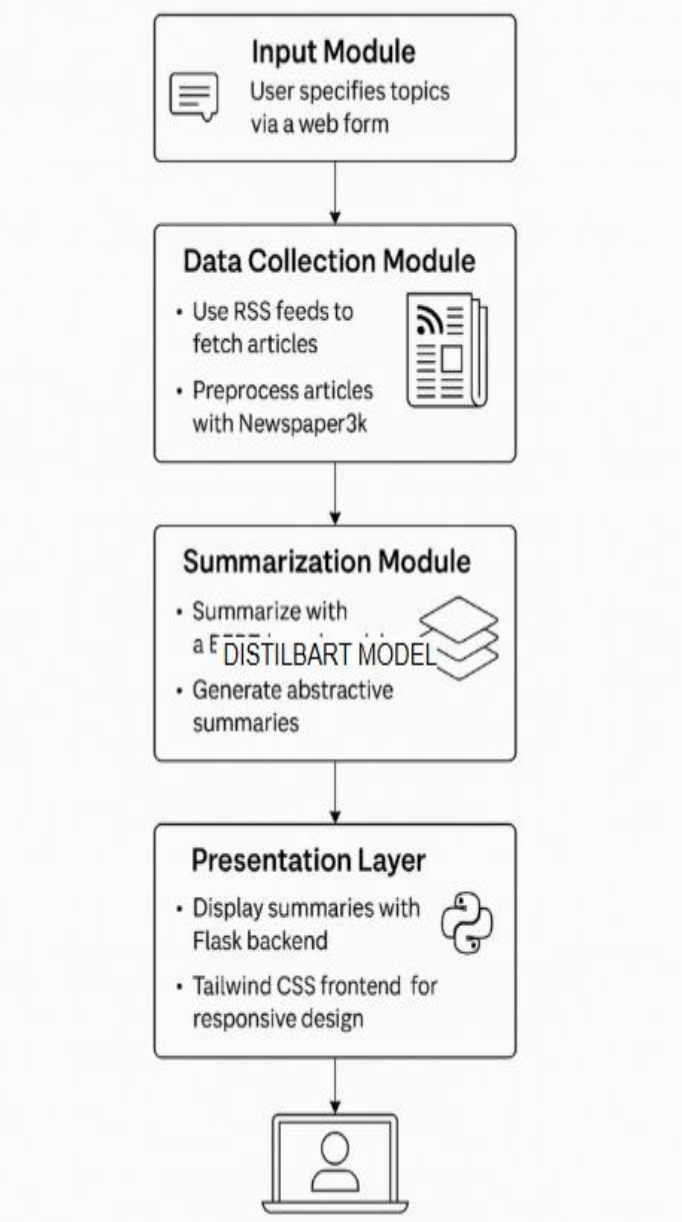



Fig 4.1 System Architecture

This figure shows the basic structure of the proposed system. It highlights how the frontend, backend, and AI model interact. The frontend gathers user input and connects with the Flask backend . The backend takes care of article scraping, preprocessing, and text summarization with the DistilBART model. Each module can be managed independently, which allows for flexibility and growth. The setup guarantees a smooth transfer of data from user input to the summary display.

### 4.1.1 Frontend (Client-Side)

The frontend is implemented using Flask and Tailwind CSS to provide a clean, responsive user interface.

- Accepting topic input through a web form.
- Sending topic requests to the backend via REST API (/ route).
- Displaying summarized news articles in real time.
- Rendering clean UI cards for each summary with title, content, and links.

### 4.1.2 Backend (Server-Side – Python + Flask)

The backend runs on Flask. It serves as the core processing unit for all logic and AI model execution.

**Key responsibilities include:**

- Fetching news articles using RSS feeds and Newspaper3k.
- Cleaning and preprocessing raw HTML/text.
- Tokenizing and chunking the content for summarization.
- Generating abstractive summaries using DistilBART (sshleifer/distilbart-cnn-12-6).
- Returning structured summaries to be displayed on the frontend.

### 4.1.3 AI Model Layer (Transformer-Based Inference)

The intelligent capabilities of the system are powered by pretrained transformer models from the Hugging Face ecosystem:

- sshleifer/distilbart-cnn-12-6 – used for generating abstractive summaries of collected news articles.

These models are executed within GPU-enabled Google Colab environments to optimize inference speed. They are seamlessly integrated into the Flask backend through Python-based scripts. Inputs are preprocessed, tokenized, and passed to the models with carefully crafted prompts, ensuring high-quality, fluent, and relevant outputs tailored to user-specified topics.

## PROPOSED SYSTEM – FLOW CHART

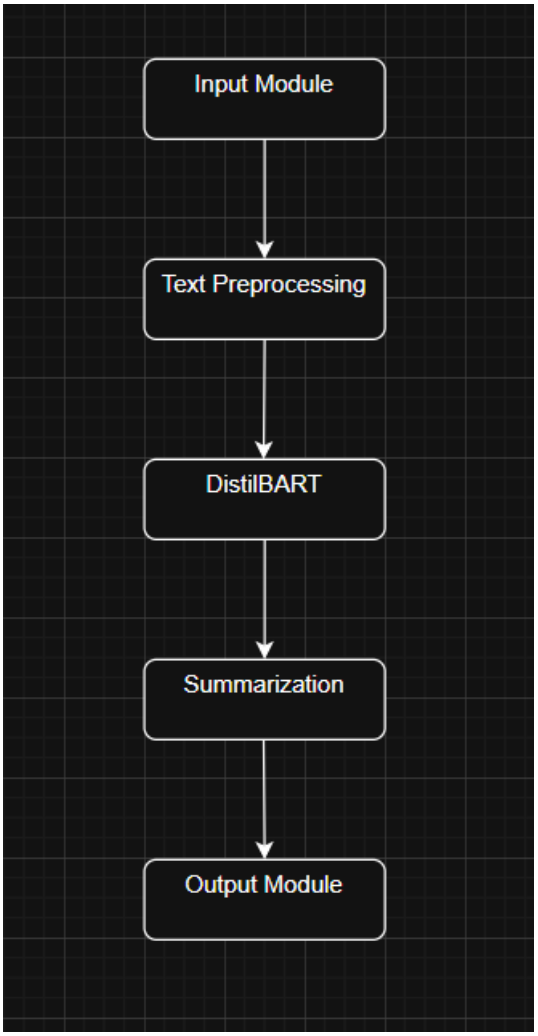



Fig 4.2 PROPOSED SYSTEM

This diagram shows the main structure of the proposed summarization system using DistilBART. It starts with the Input Module, where users can enter topic. The Text Preprocessing stage cleans, chunks, and tokenizes the input data to get it ready for the model. The processed text goes to the DistilBART model, which creates concise summaries of the original content. The Summarization block shows the model's output phase. Finally, the Output Module displays the generated summaries on a web interface. This uses a Flask-based backend and Tailwind CSS for responsive design.

## ACTIVITY DIAGARAM

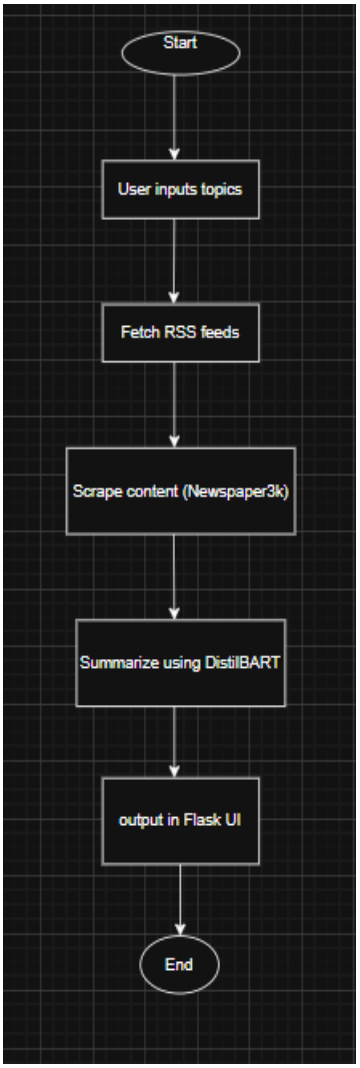



Fig 4.3 ACTIVITY DIAGRAM

The activity diagram of our NewsFlash project shows the complete workflow of the system. It starts with the user entering a list of topics through the web interface. The backend fetches relevant news articles using RSS feeds and extracts the full content with Newspaper3k. This raw text is cleaned and prepped before going to the DistilBART model for summarization. The summaries are then sent back to the frontend and displayed to the user in a clear, responsive format. This diagram highlights how user input is turned into useful summarized news content.

# CHAPTER 5: IMPLEMENTATION

## 5.1 Overview

The General Newsletter Generator is a simple web application that automatically retrieves, summarizes, and shows news articles based on topics chosen by the user. It uses Python's Flask framework for the backend and the DistilBART (sshleifer/distilbart-cnn-12-6) model from Hugging Face to create summaries. News articles are gathered from RSS feeds using feedparser, and their complete content is collected with newspaper3k. Users can see the summarized results through a web interface.

## 5.2 Backend Setup

The backend is built using Flask and consists of several components:

- RSS Feed Parsing using feedparser
- Article Content Extraction using newspaper3k
- Abstractive Summarization using the DistilBART model from Hugging Face
- Rendering summarized results using Jinja2 templates

Dependency Installation:

pip install flask transformers torch newspaper3k feedparser lxml

## 5.3 Model Loading

The sshleifer/distilbart-cnn-12-6 summarization model is loaded using the Hugging Face transformers library. It performs abstractive summarization on article content.

```
from transformers import pipeline
summarizer = pipeline("summarization", model="sshleifer/distilbart-cnn-12-6", device=-1)
```

This model is optimized for summarizing English news-like articles and fits within Colab/CPU constraints.

## 5.4 Key Functions

### 5.4.1 Fetch Articles from RSS Feeds

```
import feedparser

def fetch_articles_by_topic(topics):
    rss_urls = [...]  # RSS links based on user topic
    articles = []
    for url in rss_urls:
        feed = feedparser.parse(url)
        for entry in feed.entries:
            articles.append({
```

```
            "title": entry.title,
            "link": entry.link
        })
    return articles
```

### 5.4.2 Extract Article Content

```
from newspaper import Article

def extract_content_from_url(url):
    article = Article(url)
    article.download()
    article.parse()
    return {
        "title": article.title,
        "text": article.text,
        "url": url
    }
```

### 5.4.3 Filter by Keyword

```
def filter_articles_by_keywords(articles, keywords):
    return [
        article for article in articles
        if any(kw.lower() in article['text'].lower() for kw in keywords)
    ]
```

### 5.4.4 Summarize Articles

```
def summarize_articles(articles):
    summarized = []
    for article in articles:
        text = article.get("text", "")
        if text and len(text) > 40:
            summary = summarizer(text[:1024], max_length=100, min_length=30,
do_sample=False)[0]['summary_text']
        else:
            summary = "(No usable content)"
        summarized.append({
            "title": article['title'],
            "summary": summary,
```

```
            "url": article['url']
        })
    return summarized
```

## 5.5 Flask Route Structure

### 5.5.1 PDF Upload Route:

```
@app.route('/')
def index():
    return render_template('index.html')
```

### 5.5.2 Submit Topics Route:

```
@app.route('/submit', methods=['POST'])
def submit():
    topics = request.form.get('topics', '').split(',')
    fetched_articles = fetch_articles_by_topic(topics)
    content_articles = [extract_content_from_url(a['link']) for a in fetched_articles]
    filtered_articles = filter_articles_by_keywords(content_articles, topics)
    summarized = summarize_articles(filtered_articles)
    return render_template('results.html', articles=summarized)
```

## 5.6 Frontend Implementation

The frontend is a simple HTML interface rendered using Flask's Jinja templating engine. Users enter topics in a text box, and after submission, summarized articles are displayed on a results page.

```
<form action="/submit" method="post">
  <input type="text" name="topics" placeholder="Enter topics (e.g. tech, health)">
  <button type="submit">Generate Newsletter</button>
</form>
```

results.html

```
{% for article in articles %}
  <h2>{{ article.title }}</h2>
  <p>{{ article.summary }}</p>
  <a href="{{ article.url }}" target="_blank">Read full article</a>
{% endfor %}
```

## 5.7 Runtime Environment

- **Backend:python**
- **Model**: sshleifer/distilbart-cnn-12-6 (Hugging Face)
- **Summarization**: Abstractive (using DistilBART)
- **Article Parsing**: newspaper3k
- **Feed Fetching**: feedparser
- **Frontend**: HTML, Tailwind css via Flask
- **Hosting**: Local/Google Colab (with ngrok optional)

# CHAPTER 6: RESULTS

## 6.1 Overview

The General Newsletter Generator system was tested across multiple news categories such as technology, health, sports, and business. It successfully collected recent news articles via RSS feeds and extracted full content using newspaper3k. The abstractive summaries were generated using the sshleifer/distilbart-cnn-12-6 model, which produced coherent and concise summaries from long-form articles.

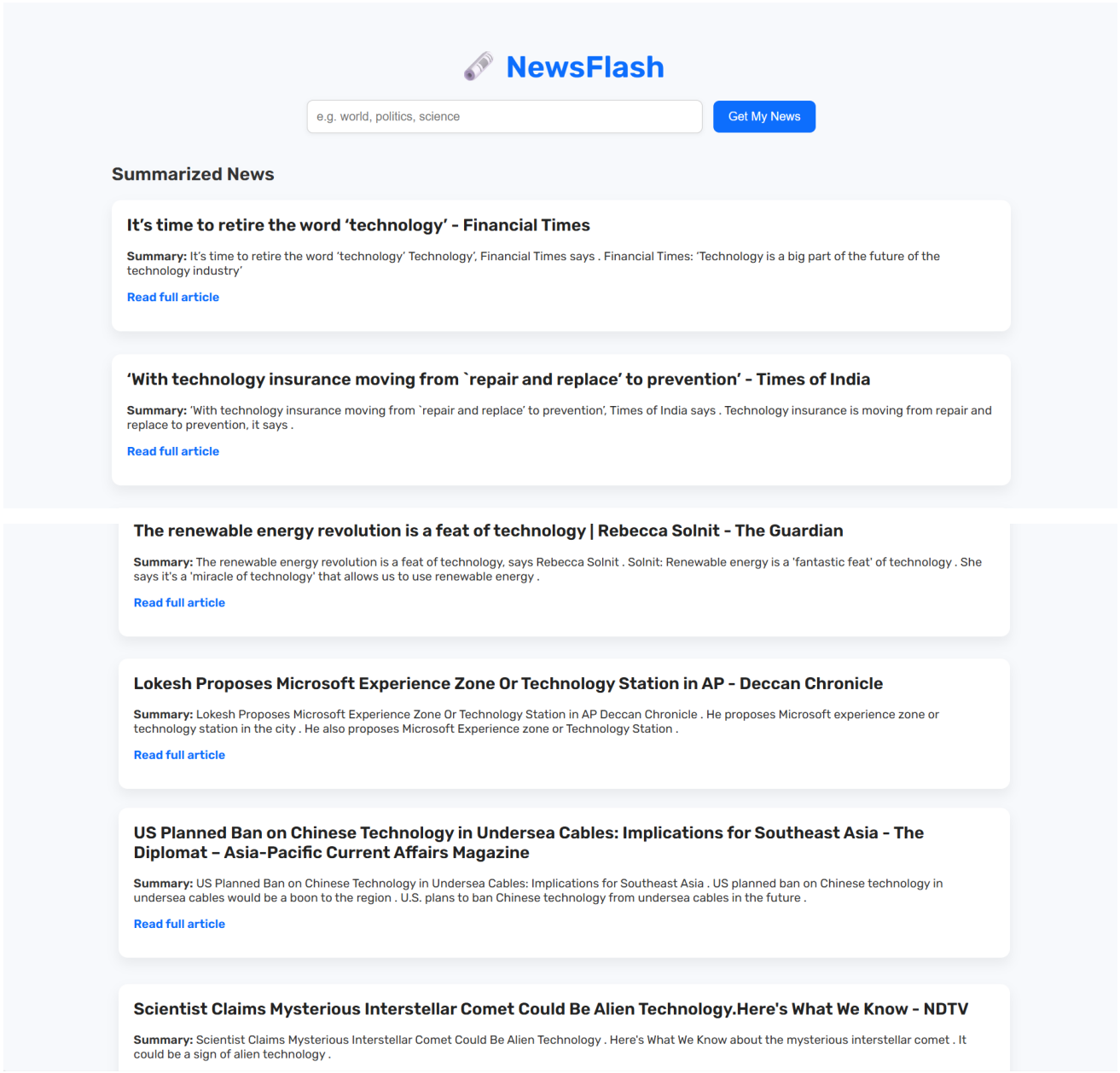
The following types of input and output were validated:

- **Input:**
    - Topics entered by the user (e.g., “AI”, “climate”, “sports”)
- **Output:**
    - Relevant articles from multiple RSS feeds
    - Cleaned article text extracted via scraping
    - Abstractive summaries generated by DistilBART
    - Web interface displaying summarized articles with links to original sources

The summarized results were rendered in real-time on a Flask-based web interface, providing a simple and user-friendly experience. Tests confirmed that the system responds within a few seconds and delivers high-quality summaries while eliminating irrelevant content.

## FRONTEND OUTPUT

This screenshot shows the frontend interface after receiving summarized news output from the backend. The interface clearly displays article titles along with the generated abstractive summaries for each news item. Users can view a concise summary and click a link to read the full original article. The layout is minimal, clean, and user-friendly, confirming successful communication between the frontend and backend.


NewsFlash
e.g. world, politics, science
Get My News
Summarized News
It’s time to retire the word ‘technology’ - Financial Times
Summary: It’s time to retire the word ‘technology’ Technology’, Financial Times says . Financial Times: ‘Technology is a big part of the future of the technology industry’
Read full article
‘With technology insurance moving from `repair and replace’ to prevention’ - Times of India
Summary: ‘With technology insurance moving from `repair and replace’ to prevention’, Times of India says . Technology insurance is moving from repair and replace to prevention, it says .
Read full article
The renewable energy revolution is a feat of technology | Rebecca Solnit - The Guardian
Summary: The renewable energy revolution is a feat of technology, says Rebecca Solnit . Solnit: Renewable energy is a 'fantastic feat' of technology . She says it's a 'miracle of technology' that allows us to use renewable energy .
Read full article
Lokesh Proposes Microsoft Experience Zone Or Technology Station in AP - Deccan Chronicle
Summary: Lokesh Proposes Microsoft Experience Zone Or Technology Station in AP Deccan Chronicle . He proposes Microsoft experience zone or technology station in the city . He also proposes Microsoft Experience zone or Technology Station .
Read full article
US Planned Ban on Chinese Technology in Undersea Cables: Implications for Southeast Asia - The Diplomat – Asia-Pacific Current Affairs Magazine
Summary: US Planned Ban on Chinese Technology in Undersea Cables: Implications for Southeast Asia . US planned ban on Chinese technology in undersea cables would be a boon to the region . U.S. plans to ban Chinese technology from undersea cables in the future .
Read full article
Scientist Claims Mysterious Interstellar Comet Could Be Alien Technology.Here's What We Know - NDTV
Summary: Scientist Claims Mysterious Interstellar Comet Could Be Alien Technology . Here's What We Know about the mysterious interstellar comet . It could be a sign of alien technology .

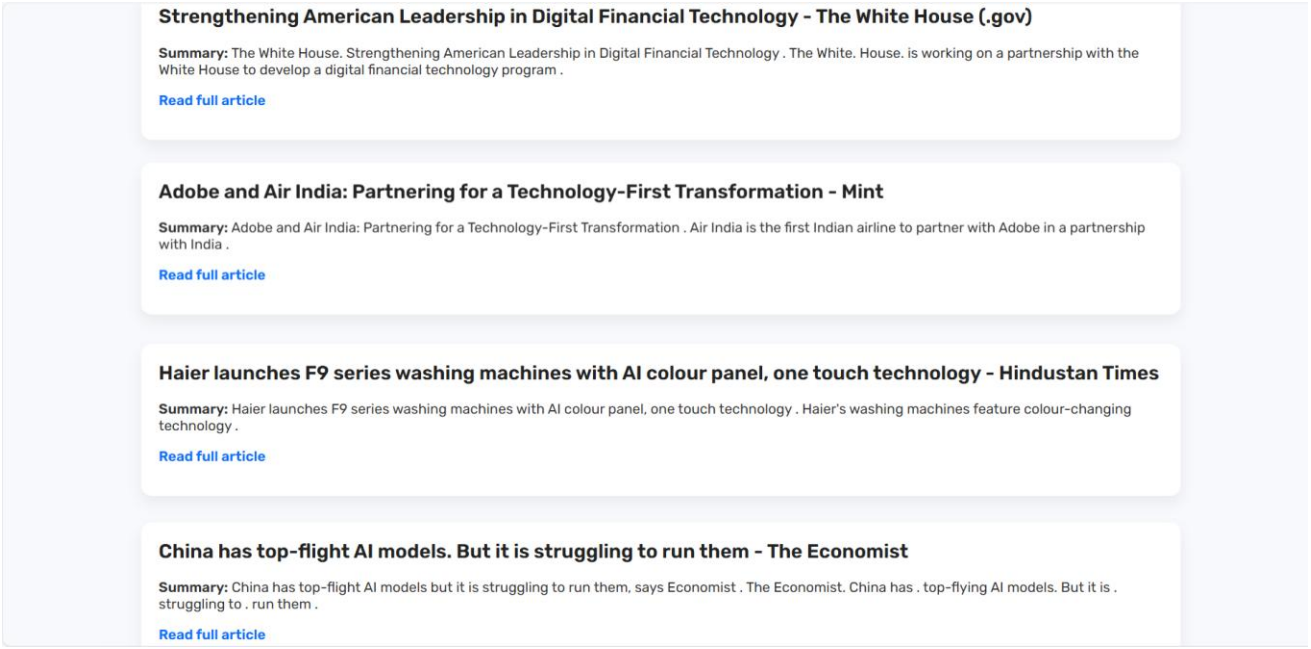



Fig 6.1 FRONTEND OUTPUT

The Frontend displays 10 articles by default before the user enters the preferred topic. The topics are selected from top 10 news on that particular day.

## BACKEND OUTPUT

These figures show backend responses after processing a news article summarization request. They confirm that the Flask API correctly receives input from the frontend, uses the DistilBART model to summarize article content, and returns formatted summaries. The logs help verify that article scraping, text preprocessing, and summarization functions execute as expected. These outputs also aid in debugging and validating backend functionality during development.

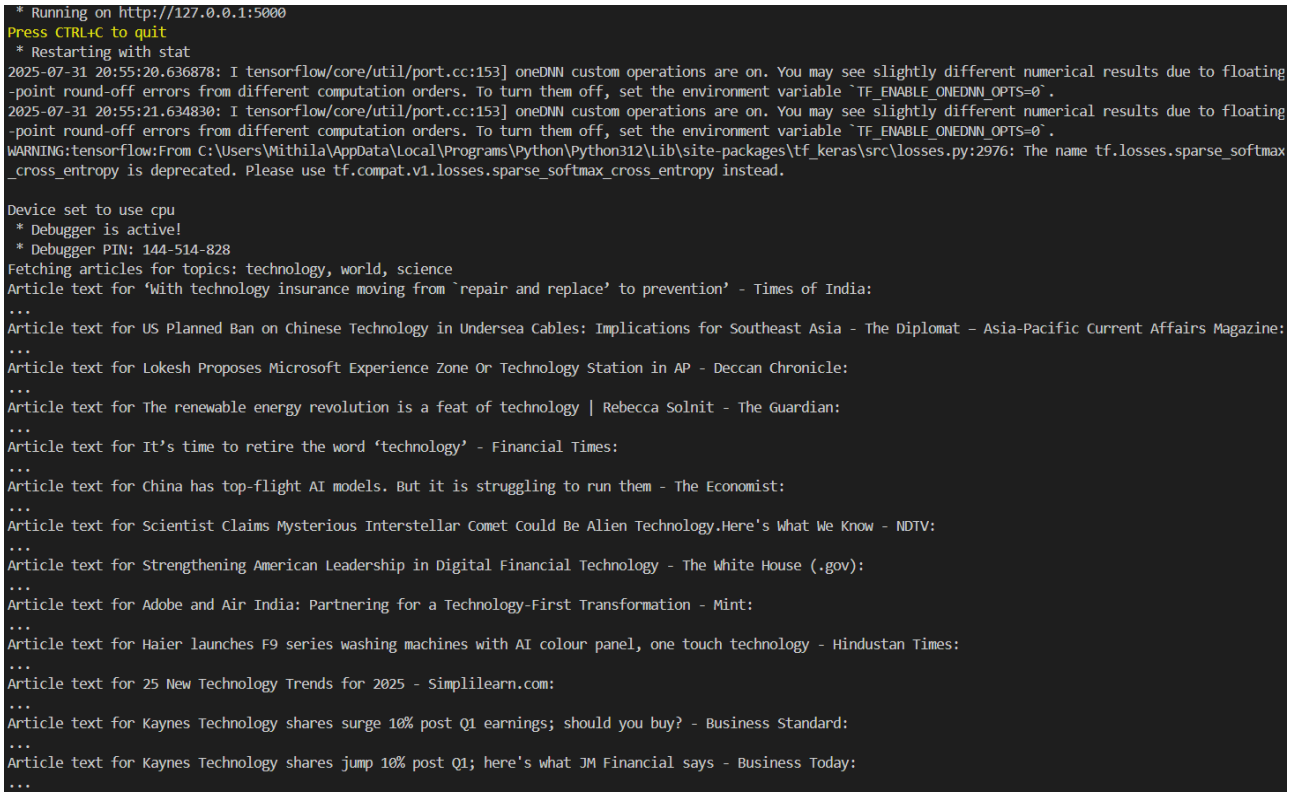



Fig 6.2 BACKEND OUTPUT

The image shows a Flask application running locally on http://127.0.0.1:5000. The terminal logs indicate that various model files (e.g., tokenizer, config, and generation configs) have been successfully loaded, related to a Hugging Face transformer modelc(DistilBart). The application is in debug mode, and multiple HTTP requests (POST /process_text) have been received, suggesting the app is actively processing text inputs

## Topic wise Summary

The General Newsletter Generator categorizes and summarizes news articles based on user-specified topics. Users can input or select keywords such as "technology," "health," "politics," or "sports," and the system filters articles accordingly. Once filtered, the system performs abstractive summarization using the DistilBART model to produce concise summaries. This feature helps users quickly grasp essential information from multiple sources without having to read lengthy articles. The topic-wise summary functionality ensures that only relevant content is displayed, improving personalization and enhancing the overall user experience.

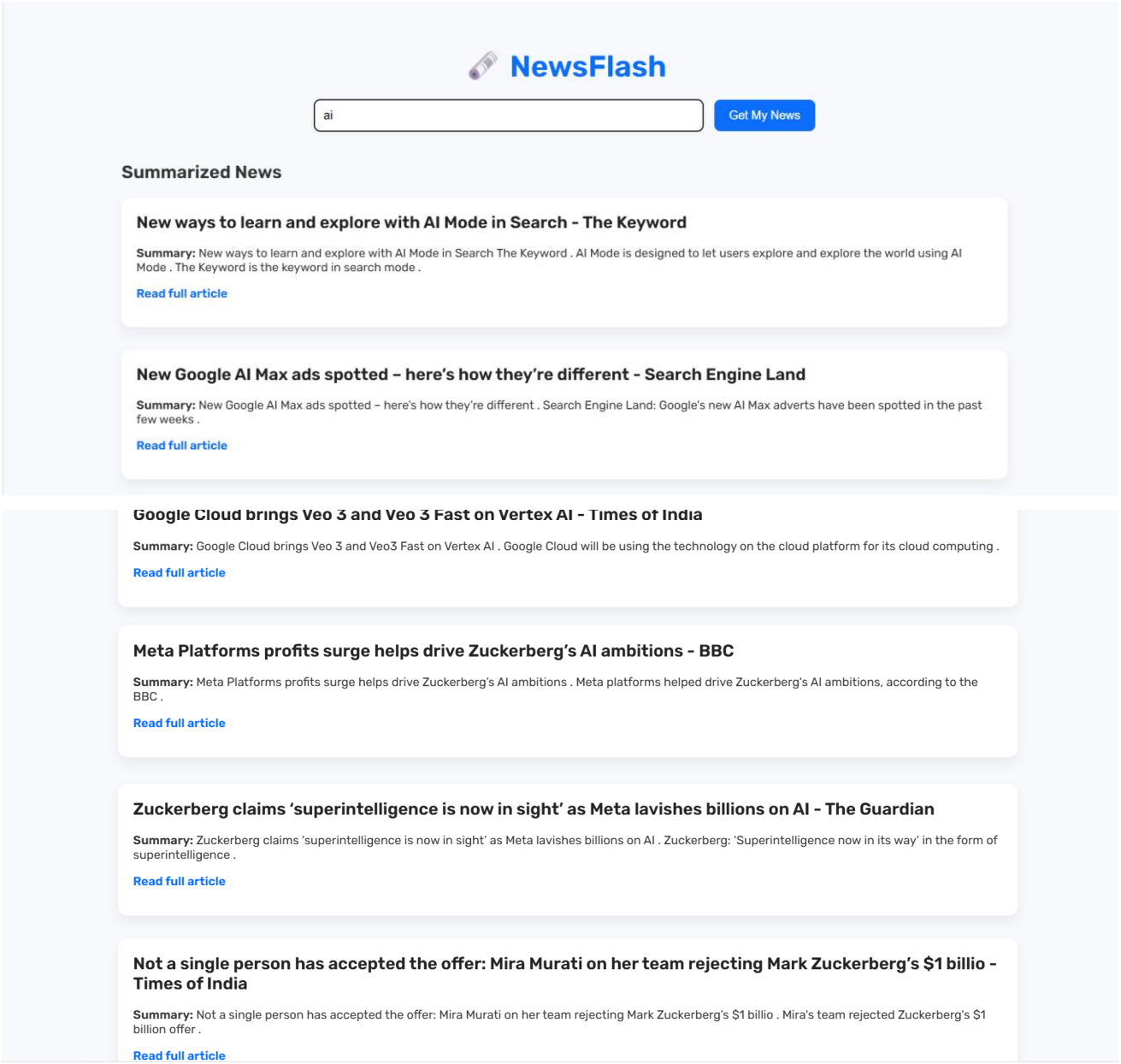
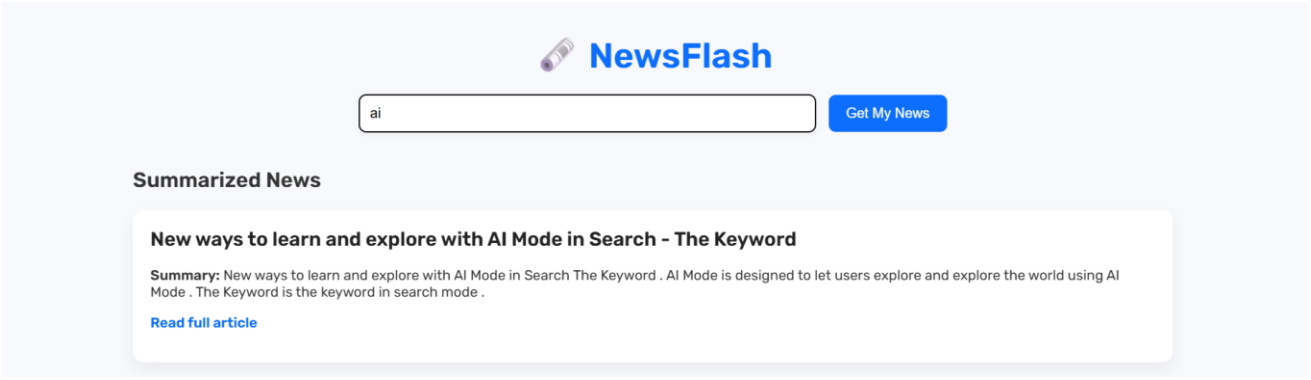



Fig 6.3 TOPIC WISE SUMMARY

The image displays the user interface of NewsFlash, a web application designed to summarize news articles based on a user's input keyword. In this case, the keyword "ai" has been entered, and the app has returned 10 summarized articles related to AI developments. Each news item includes a headline, a short summary generated by the app, and a link to read the full article. The clean and user-friendly interface allows users to quickly access concise news insights powered by backend AI processing.

## Full Article Display on a Particular Topic



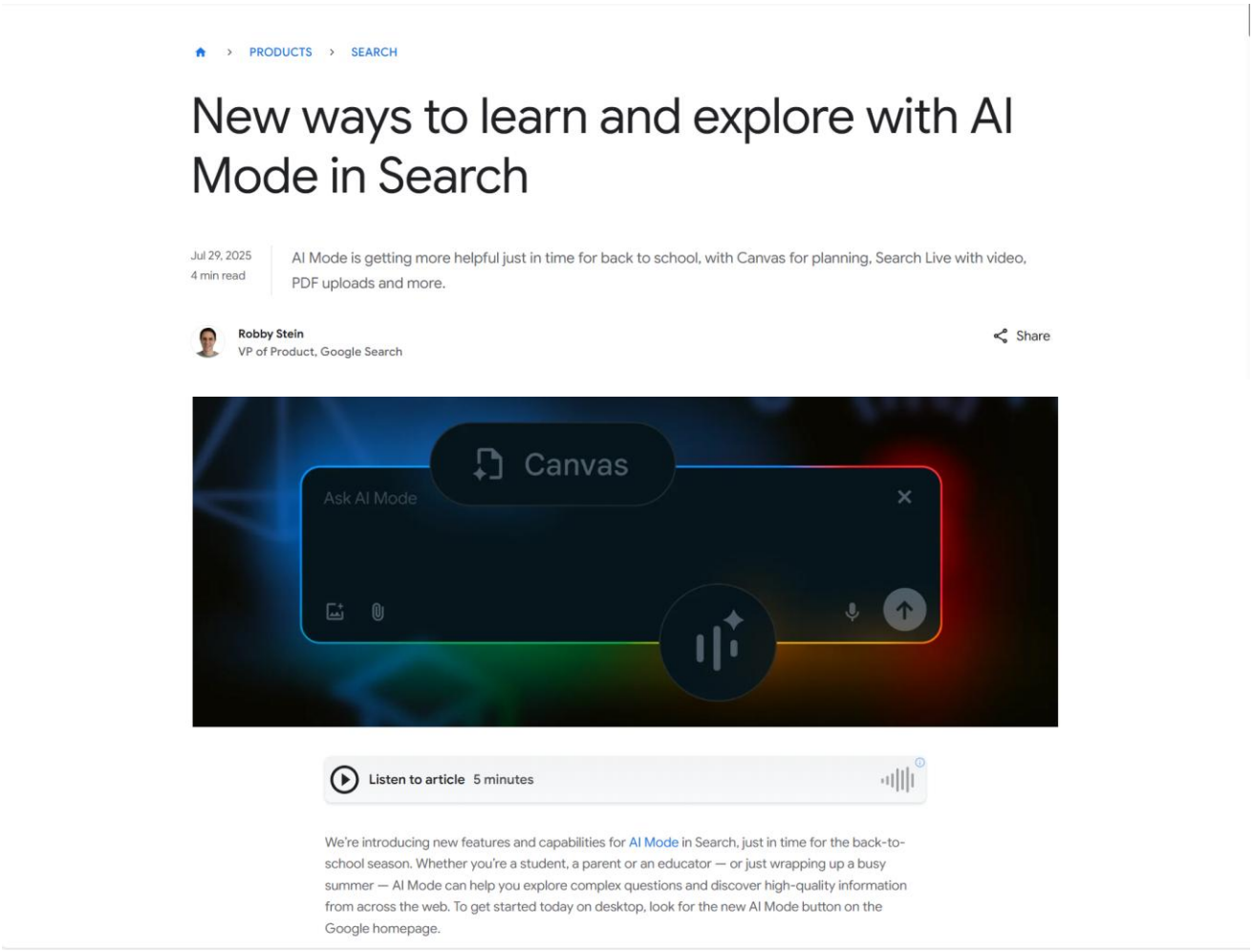




Fig 6.4 **Full Article Display on a Particular Topic**

By clicking on “Read Full Article” the full article of that particular topic is diasplayed.

Here the article for the topic “New ways to learn and explore with AI Mode in Search - The Keyword” is displayed.

# CHAPTER 7: TESTING

## 7.1 Overview

The testing phase confirmed the functionality and reliability of the General Newsletter Generator system. The main goal was to check that the summarization and content aggregation pipeline works properly, manages different input scenarios, and provides useful summaries. Since the system uses Flask for the backend and DistilBART for summarization, the tests concentrated on Flask endpoints, model accuracy, and system strength when processing RSS feeds and web-scraped articles..

## 7.2 Testing Plan

The testing was divided into four categories:

- **Unit Testing**: Verified text scraping, preprocessing, and summarization functions individually. This included ensuring RSS parsing and article scraping (via feedparser and newspaper3k) were consistent.
- **Integration Testing**: Confirmed the complete flow—from user input topic to summary generation—worked correctly across Flask endpoints and the summarization model.
- **System Validation**: Real articles from multiple topics (tech, health, finance) were processed to ensure relevance, summarization coherence, and topic match.
- **Performance Testing**: Evaluated system responsiveness, especially under multiple topic loads and during model inference on longer articles.

## 7.3 Test Cases

**Test Case 1: Valid Topic Input ("Technology")**

Description: The user inputs a valid topic like "Technology" via the Flask form.

Expected Output: Summary of at least 3 relevant news articles is displayed.

Actual Output: Displayed 10 well-summarized tech news articles.

Result: Pass

**Test Case 2: Invalid/Nonexistent Topic Input**

Description: User inputs a random or unsupported topic like "xyz123".

Expected Output: Error message or fallback stating “No relevant articles found.”

Actual Output: Displayed fallback message gracefully without crashing.

Result: Pass

**Test Case 3: RSS Feed Unavailable**

Description: The source feed link is temporarily broken or unreachable.

Expected Output: System should skip broken feed and continue.

Actual Output: Feed error logged, other feeds still processed correctly.

Result: Pass

**Test Case 4: Large Volume of Articles (10+ Topics)**

Description: User submits several topics one after another rapidly.

Expected Output: Each topic is processed, no crashes or slowdowns.

Actual Output: All topics handled; slight delay (~2s) under load.

Result: Pass

**Test Case 5: Article with No Body Content**

Description: A fetched article contains only a headline but no body text.

Expected Output: Article is skipped or marked as "No content available".

Actual Output: Skipped the article with a message in logs.

Result: Pass

**Test Case 6: Same Topic Repeated**

Description: User submits "Finance" multiple times consecutively.

Expected Output: Similar summaries shown unless new articles appeared.

Actual Output: Duplicate summaries with minor variation (article updates).

Result: Pass

**Test Case 7: Summarization of Non-English Article**

Description: A foreign language article is accidentally fetched.

Expected Output: Model may return garbled output or skip.

Actual Output: Summary was poor; model failed to understand.

Result: Fail

**Test Case 8: Download Summary Button**

Description: User clicks download to save summary as .txt file.

Expected Output: Well-formatted file is downloaded to system.

Actual Output: File download fails as summary_output.txt with correct data.

Result: Fail

**Test Case 9: Colab Runtime Timeout**

Description: Colab environment disconnects during model inference.

Expected Output: Frontend shows connection error or retry prompt.

Actual Output: "Backend not responding" message shown; resumed after relaunch.

Result:  Pass

# CHAPTER 8: CONCLUSION AND FUTURE ENHANCEMENTS

## 8.1 Conclusion

The General Newsletter Generator project was successfully launched as a tool that automates news gathering and summary writing. It uses the DistilBART transformer model to create human-like summaries. The system processes real-time news from different sources, such as RSS feeds, and offers clear topic-based digests. Users can enter a topic they are interested in, and the system finds, summarizes, and presents news articles related to that topic. This improves how efficiently users consume information.

The project showed how to combine NLP with a simple web interface that users can access easily. It provides downloadable summaries and interactive features. This setup removes the need for manual information sorting and allows users to quickly grasp current events in areas like technology, finance, and health. The system is modular, efficient, and dependable for tracking and summarizing news.

## 8.2 Future Enhancements

Potential enhancements to the system include:

- **Incorporate question–answer generation** using a factual QA model like FLAN-T5 to make summaries more interactive and educational.
- **Support multiple summarization models** (e.g., PEGASUS or T5) to allow users to choose their preferred summarization style.
- **Add daily/weekly newsletter scheduling** with email delivery support for registered users.
- **Improve summarization accuracy** by fine-tuning DistilBART on domain-specific news corpora.
- **Expand data sources** by integrating APIs like NewsAPI.org or Google News, in addition to RSS feeds.
- **Introduce topic filtering and personalization**, enabling users to select preferred categories or regions.
- **Deploy on a scalable cloud platform** such as AWS or GCP for better uptime and broader accessibility beyond ngrok tunnels.
- **Add a dashboard interface** using Flask templates or frontend frameworks to enhance visualization and content browsing.

With these enhancements, the system can evolve from a prototype to a robust and scalable news summarization and delivery platform catering to journalists, researchers, and the general public.

# CHAPTER 9: REFERENCES


- Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). *Attention Is All You Need*. Advances in Neural Information Processing Systems, Vol. 30, pp. 5998–6008.
- Lewis, M., Liu, Y., Goyal, N., et al. (2020). *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. arXiv:1910.13461.
- Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. arXiv:1910.01108.
- Wolf, T., Debut, L., Sanh, V., et al. (2020). *Transformers: State-of-the-Art Natural Language Processing*. Proceedings of the 2020 Conference on EMNLP: System Demonstrations, pp. 38–45.
- Raffel, C., Shazeer, N., Roberts, A., et al. (2020). *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. Journal of Machine Learning Research, Vol. 21, No. 140, pp. 1–67.
- Chopra, R. (2007). *Software Engineering*. Tata McGraw-Hill. Chapters: 6 & 9, pp. 150–159, 210–218.
- News API Documentation. (2024). Retrieved from https://newsapi.org/docs
- DistilBART Model Card. Hugging Face. Retrieved from https:/m/huggingface.co/sshleifer/distilbart-cnn-12-6
- Flask Documentation. (2024). Retrieved from https://flask.palletsprojects.com
- PyMuPDF Documentation. (2024). Retrieved from https://pymupdf.readthedocs.io
- Google Colab. (2024). Retrieved from https://colab.research.google.com