Multi-document Query-Based Abstractive Summarizer
Capstone Project Report

BACHELOR OF TECHNOLOGY
In
COMPUTER SCIENCE AND ENGINEERING

By
M.S.L.V.Saranya - FET/BCE/2021-25/004
Aditya Narayan Panda - FET/BCE/2021-25/005
Dibyanshu Mohapatra - FET/BCE/2021-25/008
Shreya Adya - FET/BCE/2021-25/047

Under the Guidance of
Prof. (Dr.) Gopinath Palai

FACULTY OF ENGINEERING & TECHNOLOGY
SRI SRI UNIVERSITY, CUTTACK

ABSTRACT

Effectively obtaining pertinent information from several sources in answer to a particular question has grown in importance in the age of information overload. This need is met by multi-document query-based abstractive summarizing, which uses data from several texts to create a logical, succinct, and educational summary that responds to a user-specified query. Abstractive approaches seek to produce original sentences that human-like encapsulate the information, in contrast to extractive methods that just put together phrases from source texts. With an emphasis on methods like encoder-decoder architectures, attention mechanisms, and pre-trained language models like T5 and BART, this study examines current developments in neural models for query-focused summarization. We also go over important issues including reducing repetition, factual consistency, and content relevance. Through empirical analysis on benchmark datasets, we evaluate the effectiveness of state-of-the-art models and propose enhancements to improve query alignment and summary fluency.

Sri Sri University
Faculty of Engineering & Technology

Certificate of Completion

This is to certify that the capstone project titled:
"Multi-document Query-Based Abstractive Summarizer"
has been successfully completed by M.S.L.V. Saranya (FET/BCE/2021-25/004), Aditya Narayan Panda (FET/BCE/2021-25/005), Dibyanshu Mohapatra (FET/BCE/2021-25/008) and Shreya Adya (FET/BCE/2021-25/047) under the guidance of Prof. (Dr.) Gopinath Palai, in partial fulfillment of the requirements for the award of the degree of
Bachelor of Technology in Computer Science and Engineering: Specialization in Artificial Intelligence and Machine Learning , during the academic year 2024-2025.

This project demonstrates the students' ability to apply theoretical knowledge to practical challenges, particularly in developing machine learning-driven solutions for real-world problems while also making it user friendly with interactive user interface.

Date:23-04-2025
Place: Bidyadharpur, Cuttack, Odisha

Signatures:

ACKNOWLEDGEMENT

Our profound appreciation goes out to everyone who helped us finish this project, "Multi-Document Query-Based Abstractive Summarization Using Deep Learning Techniques."

Above all, we would like to express our sincere gratitude to our project mentor, Prof. (Dr.) Gopinath Palai, for his important advice, support, and enlightening criticism during the project. His knowledge and assistance were crucial in forming the project and guaranteeing its effective completion.

We are also appreciative of Sri Sri University's Faculty of Engineering & Technology for offering the tools and a supportive atmosphere needed for research and development. Effective exploration and use of cutting-edge natural language processing techniques was made possible by the facilities and technical assistance.

Furthermore, we would like to thank a number of internet resources, such as Kaggle and other opensource platforms, for making high-quality datasets and tools available, which were essential for the creation of our estimation models.

Finally, we want to express our gratitude to our friends and family for their constant encouragement and support, which kept us going during this journey.

This project has been a great educational experience, and we are incredibly grateful to everyone who helped make it a success.

TABLE OF CONTENTS

1. INTRODUCTION

The goal of this research is to create a deep learning-based system for abstractive summarization based on multi-document queries. Users frequently find it difficult to locate succinct, pertinent responses to particular queries due to the increasing amount of textual data available online. Utilizing transformer models and sophisticated natural language processing techniques, our system synthesizes information from various sources to provide logical, human-like summaries, providing an effective information retrieval solution.

2. LITERATURE REVIEW

Recent advancements in abstractive text summarization highlight the growing interest in creating more coherent and semantically rich summaries compared to traditional extractive methods. Suleiman and Awajan (2020) presented a comprehensive review of deep learning-based abstractive summarization techniques, emphasizing the importance of RNNs, LSTMs, and attention mechanisms, while also pointing out critical challenges such as sentence repetition, out-of-vocabulary words, and factual inaccuracies. Similarly, Shakila et al. (2024) extended the scope by categorizing state-of-the-art abstractive summarization models into five families: traditional sequence-to-sequence models, pre-trained large language models, reinforcement learning, hierarchical models, and multimodal approaches. They highlighted future directions like cross-lingual summarization, factual consistency, and domain adaptation—indicating a need for models that are both semantically aware and robust across various inputs.

Focusing more narrowly on query-based summarization, Baumel et al. (2018) introduced the RSA-QFS model, a sequence-to-sequence architecture that integrates query relevance, multi-document input handling, and summary length constraints. Their work demonstrated significant improvements over extractive baselines using DUC datasets. Meanwhile, Verberne et al. (2020) explored query-based extractive summarization of discussion threads using real user queries. They found that while methods like Maximum Marginal Relevance (MMR) underperformed, combining generic post features with domain-matched word embeddings yielded more reliable summaries. However, they also emphasized the challenge of interpreting underspecified user queries and called for methods capable of generating summaries with high semantic relevance. These insights collectively inform our project direction by underlining the need for an abstractive, query-aware, multi-document summarization approach capable of balancing fluency, relevance, and efficiency.

3. OBJECTIVES

Extractive Summarization

- To create a system that can synthesize data from various textual sources to produce abstractive summaries in response to user inquiries.
- To investigate and use cutting-edge NLP and deep learning methods, especially transformer-based models, for efficient summarization.
- To guarantee that the summaries that are produced are pertinent, logical, and succinct, appropriately answering the user's question without repetition or factual mistakes.
- To make sure the system satisfies the required quality standards by assessing the summarization model's performance using benchmark datasets and standard metrics.
- To improve the information retrieval experience for users by offering a productive, automated technique for drawing insightful conclusions from massive amounts of text.

4. Methodology

This project aims to develop a **query-based, multi-document abstractive summarization system** using a fine-tuned transformer model. The methodology consists of four major components: data preprocessing, dataset transformation, model fine-tuning, and evaluation.

A. Data Preprocessing and Transformation

The dataset used for this task is derived from a structured JSON format containing documents, queries, and corresponding human-written responses. Each document may include multiple questions, each with one or more reference summaries. These were transformed into training triplets consisting of a query, context (document), and target summary. Each input was formatted as:

- Input: "summarize: <query> context: <document>"
- Target: <summary>

We filtered out empty fields, removed duplicate responses, and normalized long texts by truncating overly long inputs to fit within token limits suitable for T5 models. The processed data was converted to a .csv format and then loaded into a Hugging Face-compatible Dataset object for tokenization and training.

B. Model Fine-Tuning

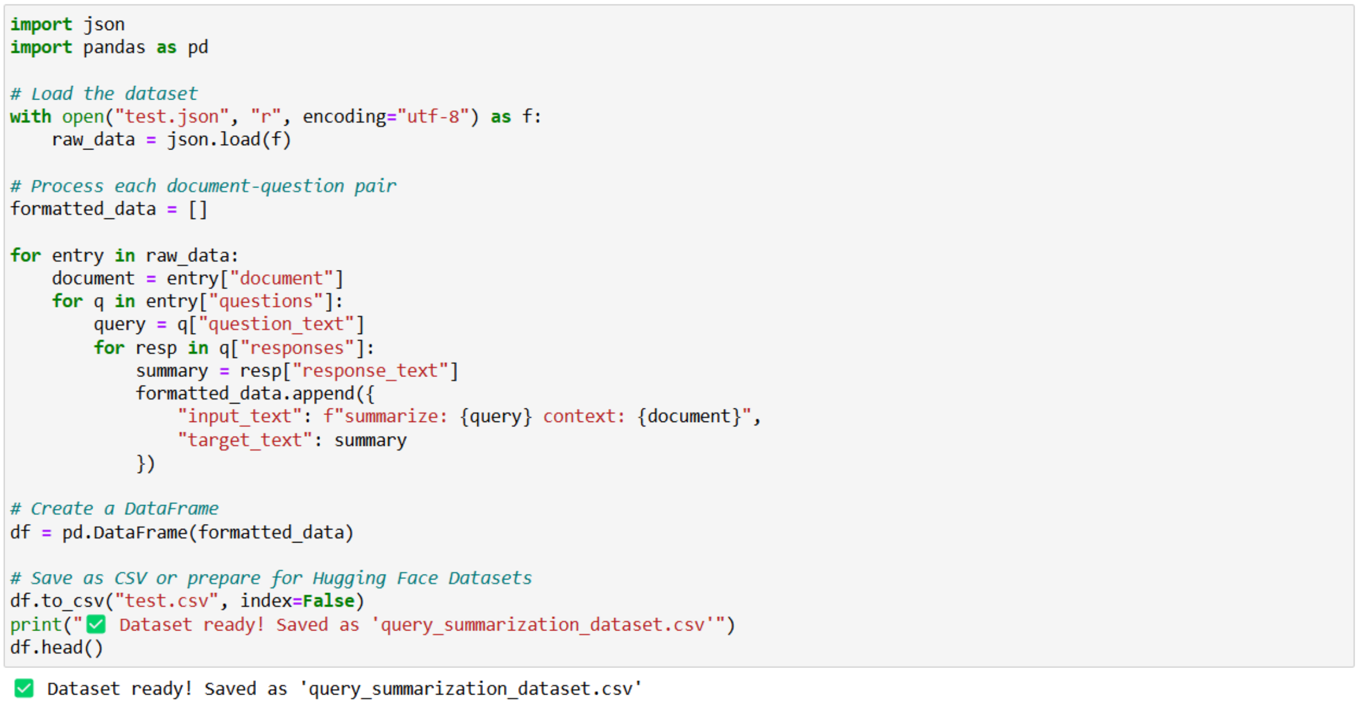
We used the t5-small model as the base for fine-tuning, leveraging Hugging Face's transformers and datasets libraries. Input and target sequences were tokenized using a maximum input length of 512 tokens and output length of 64 tokens. The model was trained using a Seq2SeqTrainer for five epochs, with early stopping, weight decay, and no-repeat n-gram constraints enabled to mitigate output repetition. Beam search with a length penalty and repetition penalty was applied during generation for more coherent summaries.

C. Evaluation

To evaluate performance, we used both **ROUGE** and **BERTScore**. ROUGE metrics were used to measure token-level overlap between generated summaries and references, while BERTScore provided semantic similarity using contextual embeddings. Although ROUGE scores were relatively low due to the abstractive nature of the summaries, BERTScore F1 scores above 0.82 indicated high semantic alignment between predictions and ground truth summaries.

5. Data Design

- Data acquired from github
- Original format - JSON Lines



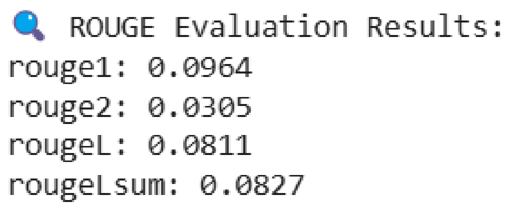
```python
import json
import pandas as pd

# Load the dataset
with open("test.json", "r", encoding="utf-8") as f:
    raw_data = json.load(f)

# Process each document-question pair
formatted_data = []

for entry in raw_data:
    document = entry["document"]
    for q in entry["questions"]:
        query = q["question_text"]
        for resp in q["responses"]:
            summary = resp["response_text"]
            formatted_data.append({
                "input_text": f"summarize: {query} context: {document}",
                "target_text": summary
            })

# Create a DataFrame
df = pd.DataFrame(formatted_data)

# Save as CSV or prepare for Hugging Face Datasets
df.to_csv("test.csv", index=False)
print("✅ Dataset ready! Saved as 'query_summarization_dataset.csv'")
df.head()
```

```
✅ Dataset ready! Saved as 'query_summarization_dataset.csv'
```

- Conversion and formatting of dataset into csv with two columns:
  - input_text (“summarize: {query} context: {document}”)
  - target_text

6. Results

- Two different metrics were used:
    - BERTScore = 82.63%
    - Rouge



```
🔍 ROUGE Evaluation Results:
rouge1: 0.0964
rouge2: 0.0305
rougeL: 0.0811
rougeLsum: 0.0827
```

Interpretation: Words and phrases used by the model don't match with Rouge's corpus.

7. Program/Code

- Model Building

```
import pandas as pd

df = pd.read_csv("query_summarization_dataset.csv")
df.head()
from datasets import Dataset
from transformers import T5Tokenizer

dataset = Dataset.from_pandas(df)
tokenizer = T5Tokenizer.from_pretrained("t5-small")

def tokenize_function(examples):
    model_inputs = tokenizer(examples["input_text"], max_length=512,
truncation=True)
    labels = tokenizer(examples["target_text"], max_length=64,
truncation=True)
    model_inputs["labels"] = labels["input_ids"]
    return model_inputs

tokenized_dataset = dataset.map(tokenize_function, batched=True)
from transformers import T5ForConditionalGeneration, Seq2SeqTrainer,
Seq2SeqTrainingArguments

model = T5ForConditionalGeneration.from_pretrained("t5-small")

training_args = Seq2SeqTrainingArguments(
    output_dir="./t5_qfs_model",
    per_device_train_batch_size=4,
    num_train_epochs=5,
    save_strategy="epoch",
    logging_dir="./logs"
)

trainer = Seq2SeqTrainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset,
    tokenizer=tokenizer
)

trainer.train()
model.save_pretrained("./qfs-t5")
tokenizer.save_pretrained("./qfs-t5")
from bert_score import score

predictions = []
references = []

for _, row in df_valid.iterrows():
    inputs = tokenizer(row["input_text"], return_tensors="pt",
truncation=True)
    output = model.generate(**inputs)
    prediction = tokenizer.decode(output[0], skip_special_tokens=True)

    predictions.append(prediction)
    references.append(row["target_text"])

# Compute BERTScore
P, R, F1 = score(predictions, references, lang="en")
print(f"\n🧠 BERTScore (F1): {F1.mean().item():.4f}")

import evaluate

rouge = evaluate.load("rouge")
# Compute ROUGE scores
rouge_scores = rouge.compute(predictions=predictions,
references=references)
print("🔍 ROUGE Evaluation Results:")
```

```
for metric, score in rouge_scores.items():
    print(f"{metric}: {score:.4f}")
```

- Flask Application
  - Structure:
    - /models
      - summarizer.py
    - /qfs-t5
      - added_tokens.json
      - config.json
      - generation_config.json
      - model.safetensors
      - special_tokens_map.json
      - spiece.model
      - tokenizer_config.json
    - /templates
      - index.html
    - /uploads
    - app.py
    - /venv

summarizer.py

```
# models/summarizer.py


from transformers import T5ForConditionalGeneration, T5Tokenizer


# Load once when the module is imported
model = T5ForConditionalGeneration.from_pretrained("./qfs-t5")
tokenizer = T5Tokenizer.from_pretrained("./qfs-t5")


def summarize_query(context, query):
  input_text = f"summarize: {query} context: {context}"
  inputs = tokenizer(input_text, return_tensors="pt", truncation=True, max_length=512)
  output = model.generate(**inputs, max_new_tokens=150)
  return tokenizer.decode(output[0], skip_special_tokens=True)
```

app.py

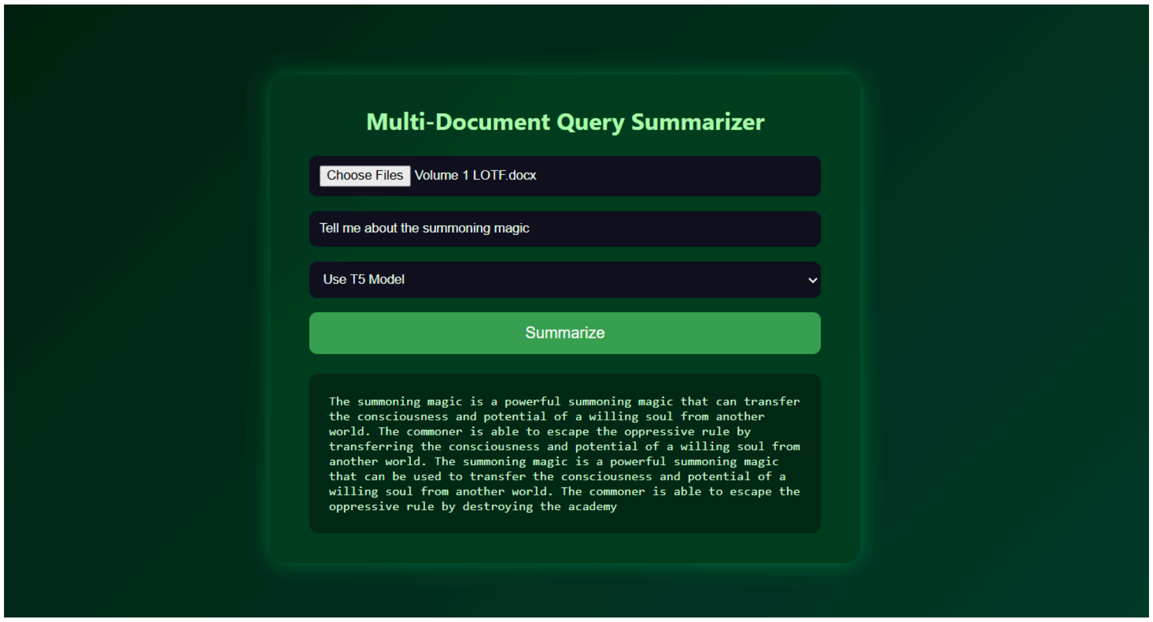
```
# app.py

from flask import Flask, request, jsonify, render_template
import os
from werkzeug.utils import secure_filename
import docx
import pdfplumber
from models.summarizer import summarize_query  # ✅ Import summarization
import subprocess
import json
from models.summarizer import summarize_query  # existing T5 summarizer

app = Flask(__name__)
UPLOAD_FOLDER = 'uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

def extract_text_from_pdf(file_path):
    with pdfplumber.open(file_path) as pdf:
        return '\n'.join([page.extract_text() or '' for page in pdf.pages])

def extract_text_from_docx(file_path):
    doc = docx.Document(file_path)
    return '\n'.join([para.text for para in doc.paragraphs])

def summarize_with_ollama(context, query):
    prompt = f"Using the following documents:\n{context}\n\nGive a summary based on the query: 
{query}"
    result = subprocess.run(
        ["ollama", "run", "tinyllama"],
        input=prompt.encode(),
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE
    )
    return result.stdout.decode('utf-8').strip()


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


@app.route('/upload', methods=['POST'])
def upload_files():
    uploaded_files = request.files.getlist("files")
    query = request.form.get("query", "")
    engine = request.form.get("engine", "t5")  # new field
    all_texts = []

    for file in uploaded_files:
        filename = secure_filename(file.filename)
        file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(file_path)

        if filename.endswith('.pdf'):
            all_texts.append(extract_text_from_pdf(file_path))
        elif filename.endswith('.docx'):
            all_texts.append(extract_text_from_docx(file_path))
    full_text = '\n'.join(all_texts)

    # Choose engine
    if engine == "ollama":
        summary = summarize_with_ollama(full_text, query)
    else:
        summary = summarize_query(full_text, query)

    return jsonify({
        "query": query,
        "engine": engine,
        "summary": summary
    })

if __name__ == '__main__':
    app.run(debug=True)
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Multi-Document Summarizer</title>
  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }

    body {
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
      background: linear-gradient(135deg, #001f0f, #003f2d);
      color: #f0fff0;
      min-height: 100vh;
      display: flex;
      align-items: center;
      justify-content: center;
      padding: 20px;
    }

    .container {
      background: rgba(0, 64, 32, 0.8);
      padding: 30px 40px;
      border-radius: 16px;
      box-shadow: 0 0 20px rgba(0, 255, 128, 0.3);
      width: 100%;
  max-width: 600px;
}

h2 {
  margin-bottom: 20px;
  text-align: center;
  color: #aefbad;
}

input[type="file"],
input[type="text"],
select {
  display: block;
  width: 100%;
  margin-bottom: 15px;
  padding: 10px;
  border: none;
  border-radius: 8px;
  background: #112;
  color: #f0fff0;
}

button {
  width: 100%;
  padding: 12px;
  background-color: #38a450;
  color: white;
  font-size: 16px;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  transition: background 0.3s;
}

button:hover {
  background-color: #2d8a44;
}

pre {
  margin-top: 20px;
  background: rgba(0, 32, 16, 0.7);
  padding: 20px;
  border-radius: 10px;
  overflow-x: auto;
  white-space: pre-wrap;
  word-wrap: break-word;
  color: #d8ffd8;
}

#loading {
  margin-top: 10px;
  font-style: italic;
  color: #aefbad;
  animation: pulse 1.5s infinite;
}

@keyframes pulse {
  0% { opacity: 0.2; }
  50% { opacity: 1; }
  100% { opacity: 0.2; }
```

```
    }
  </style>
</head>
<body>
  <div class="container">
    <h2>Multi-Document Query Summarizer</h2>
    <form id="upload-form">
      <input type="file" name="files" id="files" multiple accept=".pdf,.docx">
      <input type="text" name="query" id="query" placeholder="Enter your query here...">
      <select name="engine" id="engine">
        <option value="t5">Use T5 Model</option>
        <option value="ollama">Use Ollama (tinyllama)</option>
      </select>
      <button type="submit">Summarize</button>
    </form>
    <div id="loading" style="display: none;">🔄 Summarizing, please wait...</div>
    <pre id="output">Awaiting results...</pre>
  </div>

  <script>
    document.getElementById('upload-form').onsubmit = async function (e) {
      e.preventDefault();

      const formData = new FormData();
      const files = document.getElementById('files').files;
      const query = document.getElementById('query').value;
      const engine = document.getElementById('engine').value;

      for (let file of files) {
        formData.append('files', file);
      }

      formData.append('query', query);
      formData.append('engine', engine);

      document.getElementById('loading').style.display = 'block';
      document.getElementById('output').textContent = '';

      try {
        const response = await fetch('/upload', {
          method: 'POST',
          body: formData
        });

        const result = await response.json();
        document.getElementById('output').textContent = result.summary || "No summary returned.";
      } catch (err) {
        document.getElementById('output').textContent = 'Error: ' + err.message;
      } finally {
        document.getElementById('loading').style.display = 'none';
      }
    };
  </script>
</body>
</html>
```

8. Screenshots



Volume 1 LOTF.docx

Tell me about the summoning magic

Use T5 Model

```
The summoning magic is a powerful summoning magic that can transfer
the consciousness and potential of a willing soul from another
world. The commoner is able to escape the oppressive rule by
transferring the consciousness and potential of a willing soul from
another world. The summoning magic is a powerful summoning magic
that can be used to transfer the consciousness and potential of a
willing soul from another world. The commoner is able to escape the
oppressive rule by destroying the academy
```

9. References

1. Suleiman, D. & Awajan, A. (2020). *Deep Learning Based Abstractive Text Summarization: Approaches, Datasets, Evaluation Measures, and Challenges*. Mathematical Problems in Engineering.

2. Shakil, H., Farooq, A., & Kalita, J. (2024). *Abstractive Text Summarization: State of the Art, Challenges, and Improvements*. arXiv:2409.02413v1.

3. Baumel, T., Eyal, M., & Elhadad, M. (2018). *Query Focused Abstractive Summarization: Incorporating Query Relevance, Multi-Document Coverage, and Summary Length Constraints into Seq2Seq Models*. arXiv:1801.07704v2.

4. Verberne, S., Krahmer, E., Wubben, S., & van den Bosch, A. (2020). *Query-based Summarization of Discussion Threads*. Natural Language Engineering.