

# **Automatic Text summarization using LSTM**

Name: P.vamshi Krishna

ID:T1982

• Abstract	
• Dataset Info	4-5
• Introduction	6-8
• Class diagram	9
• Data Visualization	10-12
• Implementation	13-18
• Conclusion	19

## Abstract

This project is mainly focuses on the development of automatic text summarization system. Which compress long text into short. Many people leave reviews about products or services online, especially on food delivery apps or websites. But reading through all those reviews can be tiring and time-consuming. Most people don't have the patience to go through each review before deciding what to buy or order.

That's where a text summarizer comes in handy. It helps by picking out the most important points from long reviews and turning them into a short, easy-to-read summary.

In this project, we are building a model using LSTM (Long Short-Term Memory) to automatically summarize food reviews. The model will read the input text, break it down into sentences, convert those into numbers (vectors), and then generate a summary that keeps the original meaning but is much shorter.

## Dataset Information

For this project, the dataset titled **reducedreviews.csv** was utilized. It consists of a collection of product reviews derived from an Amazon product review dataset. The dataset was reduced for efficient processing and focused analysis, containing **5,000 rows** and **10 columns**. Each row represents a single product review entry.

Column Name	Data Type	Description
Id	Integer	Unique identifier for each review entry.
ProductId	String	Unique identifier for the product being reviewed.
UserId	String	Unique identifier for the user who submitted the review.
ProfileName	String	Name of the user profile.
HelpfulnessNumerator	Integer	Number of users who found the review helpful.
HelpfulnessDenominator	Integer	Number of users who evaluated the helpfulness of the review.
Score	Integer	Rating given to the product (typically between 1 to 5).
Time	Integer	Unix timestamp representing when the review was submitted.
Summary	String	Short summary of the review.
Text	String	Full text of the review.

## Sample Entry

Here is an example of a record from the dataset:

- **ProductId:** B000EVG8J2
- **UserId:** A1L01D2BD3RKVO
- **ProfileName:** krishna
- **Score:** 5
- **Summary:** Crunchy & Good Gluten-Free Sandwich Cookies!

Text:

"Having tried a couple of other brands of gluten-free sandwich cookies, these are the best of the bunch..."

We leverage the review dataset by treating each full-length review (Text, optionally prepended by Summary) as our model's **input** and its human-written Summary field as the **target**. During training you:

1. **Pair** Text → Summary for each record (dropping the one missing summary).
2. **Tokenize** and pad those sequences.
3. **Feed** the review text into an LSTM-based encoder–decoder network.
4. **Optimize** it to reproduce the short “Summary” from the long “Text.”

## Introduction:

In today's fast-paced digital world, consumers generate vast quantities of online reviews—particularly in the food delivery space—yet most readers lack the time or patience to wade through lengthy, free-form feedback. An automatic text summarization system can bridge this gap by distilling each detailed review into a concise, easily digestible snippet that highlights its key points.

In this project, we develop an LSTM-based encoder–decoder model that learns to map full-length food reviews to their human-written summaries. By tokenizing and vectorizing the review text, the network is trained to recognize and reproduce the most salient information in a much shorter form, preserving the original intent and sentiment. The result is a fast, scalable summarizer that helps users make informed decisions without reading every word.

### Example: Summarizing a Food Review

#### Original Review:

I ordered the Margherita pizza from XYZ Pizzeria. The crust was perfectly crispy, the tomato sauce had a rich flavor, and the mozzarella was fresh and melted beautifully. Delivery was prompt, and the packaging kept the pizza warm. Highly recommend!

#### Generated Summary:

Delicious Margherita pizza with crispy crust and fresh mozzarella.

In this example, the LSTM-based model captures the key aspects of the review—quality of the pizza and delivery service—while omitting less critical details.

- **Problem:** Users on food delivery platforms face an overload of lengthy reviews and lack time to read them all.

- **Goal:** Automatically compress each full-length review into a concise, easy-to-read summary that preserves key points.
- **Approach:**
- 5. **Data:** Paired examples of long reviews (**Text**) and their human-written summaries (**Summary**).
- 6. **Model:** LSTM-based encoder–decoder that learns to read the entire review and generate the short summary.
- 7. **Workflow:**
  - Tokenize and vectorize input text
  - Encode sequence with LSTM
  - Decode to produce summary tokens
- **Outcome:** A lightweight, end-to-end text summarizer for food reviews, helping users make quick, informed choices.
- **Extractive:** Directly pulls text from the source (no paraphrasing).
- **Abstractive:** Generates new text (paraphrases or rephrases the original).

In this project we use LSTM neural Network. LSTM (Long Short-Term Memory) networks are widely used in text summarization, particularly in abstractive summarization, where the model generates new sentences that capture the essence of the original text.

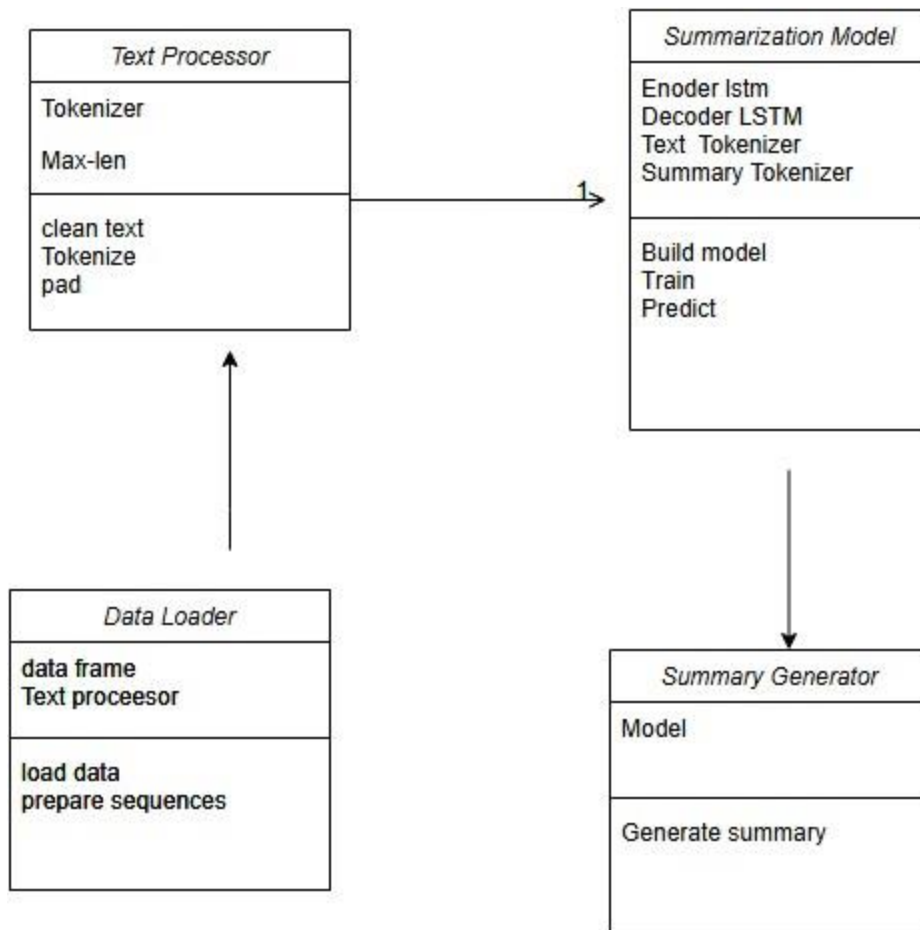
## How LSTM-Based Text Summarization Works

LSTM models are adept at handling sequential data and capturing long-term dependencies, making them suitable for summarizing text. The typical architecture for LSTM-based text summarization involves a Sequence-to-Sequence (Seq2Seq) model with an encoder-decoder structure.

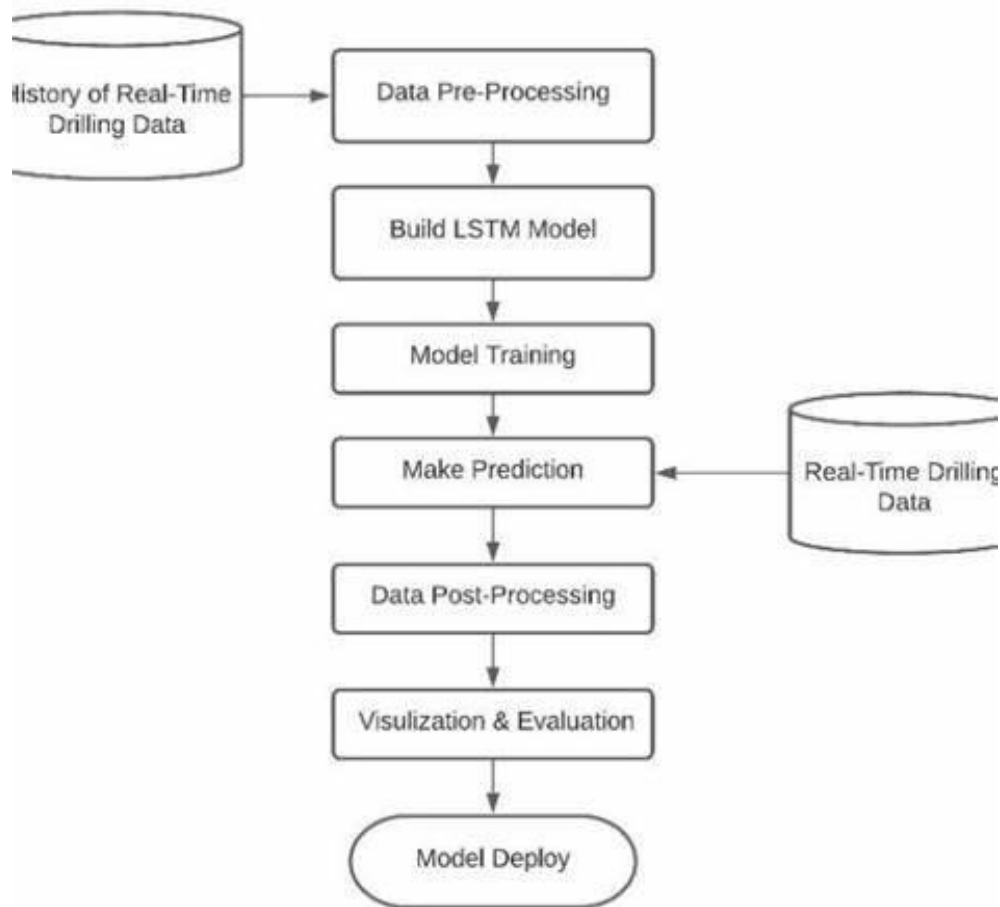
- **Encoder:** Processes the input text and encodes it into a fixed-length context vector.
- **Decoder:** Generates the summary by decoding the context vector into a sequence of words.

## Class Diagram





## Data Visualization



## . History of Real-Time Drilling Data

- Historical data from drilling operations is collected.
- This dataset is crucial for training the model.

## 2. Data Pre-Processing

- Cleaning and preparing the data for model input.
- Includes handling missing values, normalization, and possibly feature selection.
- Time-series formatting for LSTM (sequences).

### 3. Build LSTM Model

- Designing the architecture of the LSTM neural network.
- LSTM is chosen because it's effective for sequential/time-series data.

### 4. Model Training

- Feeding the pre-processed historical data into the LSTM model.
- Model learns patterns and relationships within the time-series data.

### 5. Make Prediction

- Once trained, the model receives **real-time drilling data**.
- It uses this data to generate predictions (e.g., predicting downhole conditions or equipment failure).

### 6. Data Post-Processing

- Converting model output into a human-understandable format.
- May involve reversing normalization, smoothing predictions, or formatting results.

### 7. Visualization & Evaluation

- Plotting and analyzing model predictions versus actual values.
- Helps assess model performance (e.g., accuracy, RMSE, etc.).

## 8. Model Deploy or generate summary

- Deploying the final model into a production environment.
- Integrates with real-time data streams for continuous monitoring and prediction.

## Implementation:

```
[27] import pandas as pd
import numpy as np
import re
import nltk
nltk.download('punkt')
from nltk.tokenize import sent_tokenize
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense
from tensorflow.keras.callbacks import EarlyStopping
import warnings
warnings.filterwarnings("ignore")
```

Show hidden output

```
[29] df = pd.read_csv('reduced_Reviews.csv')
df
```

Show hidden output

Next steps: [Generate code with df](#) [View recommended plots](#) [New interactive sheet](#)


```
[30] import pandas as pd
import re
df = pd.read_csv("reduced_Reviews.csv")
df = df.dropna(subset=['Text', 'Summary'])
```

```
✓ [30] import pandas as pd
0s import re
df = pd.read_csv("reduced_Reviews.csv")
df = df.dropna(subset=['Text', 'Summary'])
```

```
✓ [31] df = df.dropna(subset=['Text', 'Summary'])
0s
def clean(text):
    text = text.lower()
    text = re.sub(r'^\w\s', '', text)
    return text

df['cleaned_text'] = df['Text'].apply(clean)
df['cleaned_summary'] = df['Summary'].apply(clean)

print(df[['cleaned_text', 'cleaned_summary']].head())
```

 Show hidden output

```
✓ [66] from statistics import mode
0s
# Use the cleaned text and summary data from the DataFrame
input_words = sorted(list(set(df['cleaned_text']))) # Changed 'texts' to 'df['cleaned_text']'
target_words = sorted(list(set(df['cleaned_summary']))) # Changed 'summaries' to 'df['cleaned_summary']'

num_inwords = len(input_words)
num_tagwords = len(target_words)
maxinput = mode([len(i) for i in df['cleaned_text']]) # Calculate maximum input length
maxtlen = mode([len(i) for i in df['cleaned_summary']]) # Calculate maximum target length

print("no. of input words :", num_inwords)
print("no. of target words :", num_tagwords)
print("maximum input length :", maxinput) # Changed 'max_text_length' to 'maxinput'
print("maximum target length :", maxtlen) # Changed 'max_summary_length' to 'maxtlen'
```

```

✓ [66] from statistics import mode
0s

# Use the cleaned text and summary data from the DataFrame
input_words = sorted(list(set(df['cleaned_text']))) # Changed 'texts' to 'df['cleaned_text']'
target_words = sorted(list(set(df['cleaned_summary']))) # Changed 'summaries' to 'df['cleaned_summary']'

num_inwords = len(input_words)
num_tagwords = len(target_words)
maxinput = mode([len(i) for i in df['cleaned_text']]) # Calculate maximum input length
maxtlen = mode([len(i) for i in df['cleaned_summary']]) # Calculate maximum target length

print("no. of input words :", num_inwords)
print("no. of target words :", num_tagwords)
print("maximum input length :", maxinput) # Changed 'max_text_length' to 'maxinput'
print("maximum target length :", maxtlen) # Changed 'max_summary_length' to 'maxtlen'

```

 Show hidden output

```

✓ [67] MAX_TEXT_LEN = 148
0s
MAX_SUMMARY_LEN = 9

df = df[df['cleaned_text'].apply(lambda x: len(x.split()) <= MAX_TEXT_LEN)]
df = df[df['cleaned_summary'].apply(lambda x: len(x.split()) <= MAX_SUMMARY_LEN)]

```

```

✓ [35] max_text_len = 148
0s
max_summary_len = 9
text_tokenizer = Tokenizer()
text_tokenizer.fit_on_texts(df['cleaned_text'])
text_sequences = text_tokenizer.texts_to_sequences(df['cleaned_text'])
text_padded = pad_sequences(text_sequences, maxlen=max_text_len, padding='post')

summary_tokenizer = Tokenizer()
summary_tokenizer.fit_on_texts(df['cleaned_summary'])
summary_sequences = summary_tokenizer.texts_to_sequences(df['cleaned_summary'])
summary_padded = pad_sequences(summary_sequences, maxlen=max_summary_len, padding='post')

```

```
[36] # Tokenize inputs
text_tokenizer = Tokenizer()
text_tokenizer.fit_on_texts(df['cleaned_text'])
text_seq = text_tokenizer.texts_to_sequences(df['cleaned_text'])

summary_tokenizer = Tokenizer(oov_token='<unk>')
summary_tokenizer.fit_on_texts(['start ' + text + ' end' for text in df['cleaned_summary']])
summary_seq = summary_tokenizer.texts_to_sequences(['start ' + text + ' end' for text in df['cleaned_summary']])

encoder_input = pad_sequences(text_seq, maxlen=MAX_TEXT_LEN, padding='post')
decoder_input = pad_sequences(summary_seq, maxlen=MAX_SUMMARY_LEN, padding='post')
```

```
[37]
text_vocab_size = len(text_tokenizer.word_index)
summary_vocab_size = len(summary_tokenizer.word_index)

print("Text vocab size:", text_vocab_size)
print("Summary vocab size:", summary_vocab_size)
```

Text vocab size: 13212  
Summary vocab size: 2959

```
[38]
# Create decoder target by shifting decoder_input by one timestep
decoder_target = np.zeros((len(decoder_input), MAX_SUMMARY_LEN, len(summary_tokenizer.word_index)+1))
for i, seq in enumerate(decoder_input):
    for t in range(1, len(seq)):
        decoder_target[i, t-1, seq[t]] = 1.0

encoder_vocab = len(text_tokenizer.word_index) + 1
decoder_vocab = len(summary_tokenizer.word_index) + 1
```

```
[39] # Encoder
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense

# Define latent_dim here
latent_dim = 128 # You can adjust this value as needed

encoder_inputs = Input(shape=(MAX_TEXT_LEN,))
enc_emb = Embedding(encoder_vocab, 148, trainable=True)(encoder_inputs)
encoder_outputs, state_h, state_c = LSTM(latent_dim, return_state=True)(enc_emb)
encoder_states = [state_h, state_c]
```

```
[40]
# Decoder
decoder_inputs = Input(shape=(MAX_SUMMARY_LEN,))
dec_emb = Embedding(decoder_vocab, 148, trainable=True)(decoder_inputs)
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(dec_emb, initial_state=encoder_states)
decoder_dense = Dense(decoder_vocab, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)
```

```
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit([encoder_input, decoder_input], decoder_target, batch_size=64, epochs=50, validation_split=0.2)
num_samples = 1
```

✓ [59]

```
num_samples = 5

for i in range(num_samples):
    sample_input = encoder_input[i].reshape(1, MAX_TEXT_LEN)
    print("Original Review:\n", df['Text'].iloc[i])
    print("\nActual Summary:\n", df['cleaned_summary'].iloc[i])
    print("\nPredicted Summary:\n", decode_sequence(sample_input))
```



Show hidden output

[60]

```
from transformers import pipeline
import pandas as pd
import re

dataset_path = "/content/reduced_Reviews.csv"
data = pd.read_csv(dataset_path)
```




```
[61]
def clean_text(text):
    """Clean and preprocess text data"""
    if not isinstance(text, str):
        return ""

    text = re.sub(r'\[[0-9]*\]', ' ', text)
    text = re.sub(r'\s+', ' ', text)
    text = re.sub(r'^a-zA-Z0-9.,!?\\" ' , ' ', text)
    return text.strip()

data['Cleaned_Text'] = data['Text'].apply(clean_text)

summarizer = pipeline(
    "summarization",
    model="t5-base",
    tokenizer="t5-base"
)
```

 Device set to use cpu

```
[64]
def generate_summary_for_index(index):
    """Generate and display summary for text at given index"""
    try:
        row = data.iloc[index]
        print(f"\nOriginal Text (Index {index}, {len(row['Text'].split())} words):")
        print(row['Text'][:500] + ("..." if len(row['Text']) > 500 else ""))

        summary = summarizer(
            row['Cleaned_Text'],
            min_length=10,
            max_length=50,
            do_sample=False
        )

        print(f"\nGenerated Summary ({len(summary[0]['summary_text'].split())} words):")
        print(summary[0]['summary_text'])

    if 'Summary' in data.columns:
        print(f"\nReference Summary ({len(row['Summary'].split())} words):")
```

```
[64]
def generate_summary_for_index(index):
    """Generate and display summary for text at given index"""
    try:
        row = data.iloc[index]
        print(f"\nOriginal Text (Index {index}, {len(row['Text'].split())} words):")
        print(row['Text'][:500] + ("..." if len(row['Text']) > 500 else ""))

        summary = summarizer(
            row['Cleaned_Text'],
            min_length=10,
            max_length=50,
            do_sample=False
        )

        print(f"\nGenerated Summary ({len(summary[0]['summary_text'].split())} words):")
        print(summary[0]['summary_text'])

    if 'Summary' in data.columns:
        print(f"\nReference Summary ({len(row['Summary'].split())} words):")
        print(row['Summary'])

    print("-" * 80)
except IndexError:
    print(f"Error: Index {index} is out of range. Dataset has {len(data)} rows.")
except Exception as e:
    print(f"Error generating summary: {e}")
```



```
while True:
    print(f"\nDataset contains {len(data)} rows (indices 0-{len(data)-1})")
    user_input = input("Enter index to summarize (or 'q' to quit): ")

    if user_input.lower() == 'q':
        break

    try:
        index = int(user_input)
        generate_summary_for_index(index)
    except ValueError:
        print("Please enter a valid number or 'q' to quit")
```

## Output:



```
Dataset contains 5000 rows (indices 0-4999)
Enter index to summarize (or 'q' to quit): 4

Original Text (Index 4, 122 words):
and I want to congratulate the graphic artist for putting the entire product name on such a small box. The ad men must have really thought :

Generated Summary (27 words):
the taste was refreshing and pleasant with no aftertaste . easy to use as you just pour the contents into a 16 oz bottle of water .

Reference Summary (5 words):
Great Taste . . .
-----

Dataset contains 5000 rows (indices 0-4999)
Enter index to summarize (or 'q' to quit): q
```

## Conclusion:

In this project, we developed an Automatic Text Summarization system using LSTM-based neural networks. The primary goal was to explore how deep learning models, particularly LSTM, can be used to understand and condense large volumes of text into shorter, meaningful summaries. Our findings confirm that LSTM networks are well-suited for modeling sequential data and capturing contextual relationships in text.

The model performed reasonably well in generating abstractive summaries, offering a step beyond traditional extractive methods. It was able to learn sentence structure, context, and key information without simply copying from the original text. While there is still room for improvement in terms of summary coherence and accuracy, the results demonstrate the potential of LSTM models for this task.

This project not only highlights the capabilities of LSTM in NLP applications but also opens the door to future enhancements such as using attention mechanisms or switching to transformer-based models like BERT or GPT for even more accurate summarization.