THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

EXPLORING ABSTRACTIVE TEXT SUMMARIZATION FOR DIALOGUES USING BART

Lok Yee Joey Cheung

School of Information Technology and Electrical Engineering,
University of Queensland

25th October 2024

Abstract

Effective dialogue summarization plays a crucial role in natural language processing (NLP). It addresses the challenges of information overload and enhances communication efficiency in various applications, such as customer service and user research. Despite technological advancements, summarizing dialogues remains complicated due to the informal language and structure of dialogues, as well as the limited benchmarks for evaluation. In this project, I fine-tuned the Bidirectional and Auto-Regressive Transformers (BART) using the Samsung Abstractive Messenger Summarization (SAMSum) dataset which employed transfer learning for optimal model performance. Furthermore, I compared and contrasted traditional Recurrent Neural Networks (RNNs) with the modern Transformer-based BART model on abstractive text summarization. This project aims to provide valuable insights into enhancing dialogue summarization techniques in NLP.

The report is structured as follows: I will outline the problems and motivations, followed by a literature review of existing approaches to dialogue summarization. Next, I detail the exploratory data analysis and preprocessing steps, discuss the project preliminaries, and provide implementation details for the RNN and Transformer models. Finally, I present the results and insights from the final model, including both qualitative and quantitative analyses. My proposed solution has demonstrated excellent ability in summarizing both articles and dialogues using advanced fine-tuning technique.

Executive Summary

In the field of text summarization, abstractive methods have been a promising approach for generating coherent and meaningful summaries by creating new sentences that capture the core meaning of the text (Cohan et al., 2018). Traditionally, researchers use sequence-to-sequence models that rely on RNNs to capture temporal dependencies. However, they often struggle with long-range dependencies and lack contextual understanding (Bahdanau et al., 2014; Cho et al., 2014). Recently, self-attention mechanisms have driven innovation in text generation by allowing models to capture long-term relationships between words and generate abstractive summaries (Vaswani et al., 2017). Transformer-based language models, such as BART use self-attention and large-scale pretraining to enhance the quality of summaries. It excels in text generation tasks and offers a strong foundation for abstractive summarization.

BART is a unique generative model that combines a bidirectional encoder with an autoregressive decoder for text summarization (Lewis et al., 2019). As a denoising autoencoder, BART was pre-trained to reconstruct corrupted text by learning complex relationships between words which enhanced its generative ability across various scenarios (Lewis et al., 2019). Despite BART's potential, its pre-trained version is only optimized for structured data, namely news articles. This presents both challenges and opportunities for advancement in other domains, like dialogue summarization.

My primary objective is to extend BART's capabilities to summarize dialogues beyond the scope of its pre-trained news and journal datasets. To achieve this, I explored the approach of fine-tuning BART on high-quality, dialogue-specific dataset, which enabled domain-specific transfer learning. After the model was implemented, I evaluated and tested the effectiveness using ROUGE scores. I also manually examined the coherence in summaries generated from multi-turn conversations. By focusing on these aspects, I hope to provide a framework for the future advancement for dialogue summarization. The findings will contribute to the broader field of abstractive summarization and its applications in diverse text genres.

Table of Content

1 Introduction

In these days and ages, dialogue-based communication has been commonly used, ranging from customer service interaction to UX research interviews etc. Dialogues differ from speech as they are textual interactions in interviews, chat messages, emails, forum discussions and more (Gliwa et al., 2019). Extracting meaningful insights from these interactions is important, yet manually summarizaing dialogues can be inefficient and prone to human errors. This project focuses on automating this process by using pre-trained large language model, BART, for dialogue summarization. BART is a powerful pre-trained model used for summarizaing structured text like news articles and journals (Lewis et al., 2019). The aim of this project is to fine-tune the model to better handle conversational data and to improve the accuracy and coherence of dialogue summaries.

1.1 Problem Statement

Several problems were identified including the informal nature of dialogues and the limitation of current BART model. The nature of dialogue data presents unprecedented challenges since it includes fragmented sentences, informal language, typos, and ambiguity in contextual meaning. Unlike structured texts, dialogues can also shift rapidly between topics and tones. This variability can lead to ambiguous statements where the meaning is not explicit and clear enough, which requires a deeper understanding of context to accurately capture intent. Furthermore, problems like interruptions between speakers which result in confusion are common. However, manually analyse, digest and annotate summaries for dialogue can be time-consuming and require significant human efforts. These makes the use of large language models like BART necessary to generate coherent and contextually relevant summaries of conversational content automatically.

While the BART model is powerful for general text summarization of news article and journals, it has limitations when applied to summarizing dialogues (Lewis et al., 2019). Current BART model sometimes struggles with understanding the conversational context, capturing the key points and maintaining coherence. It may generate summaries that omit important details from a dialogue or introduce irrelevant information. Figure 1 shows an example of how the current pre-trained BART model falls short in summarizing dialogues. It inaccurately introduces new information like “a few days ago" and it incorrectly interprets the conversation from a first-person perspective. The pronouns of “I” and “we” are unclear which causes confusion. This is where fine-tuning the model on dialogue-specific datasets becomes essential. It allows BART to adapt to the structure and linguistic features of dialogues and hence creating more accurate and contextually relevant summaries (Han et al., 2021).

Input Dialogue 1

Hannah: Hey, do you have Betty's number?
Amanda: Lemme check
Hannah: <file_gif>
Amanda: Sorry, can't find it.
Amanda: Ask Larry
Amanda: He called her last time we were at the park together
Hannah: I don't know him well
Hannah: <file_gif>
Amanda: Don't be shy, he's very nice
Hannah: If you say so..
Hannah: I'd rather you texted him
Amanda: Just text him 🙂
Hannah: Urgh.. Alright
Hannah: Bye
Amanda: Bye bye

Pre-trained BART Model's
Generated Summary

A few days ago, I was talking to my friend Hannah, and we were looking for her friend Betty.

Figure 1. Example of current BART Model's dialogue summarization capability

1.2 Motivation and Methodology

There is a growing need to improve the efficiency and accuracy of dialogue summarization to mitigate the above-mentioned problems. I have addressed the following key questions in this report in order to improve the effectiveness in dialogue summarization:

1. How can the accuracy of automated dialogue summarization be improved to handle the complexities of conversational data?
2. What are the limitations of traditional and modern summarization models in summarizing dialogue, and how can they be addressed?

Dialogues are of great importance in various fields, such as journalism, UX research, interviews, usability testing, customer service, legal transcriptions etc, where extracting key insights from dialogue is crucial for qualitative analysis and decision making. For instance, in legal contexts, summarizing conversations with precision is crutial for creating reliable records as references for important decisions in courts. Similarly, applying dialogue summarization in UX research helps increase the efficiency in analyzing user feedback, identifying pain points and areas for improvement. The manual process of analyzing interviews or transcripts, however, remains labor-intensive and prone to human error.

As a result, fine-tuning BART for dialogue summarization would allow professionals in these fields to streamline their workflows, reduce human error, and increase the overall efficiency and effectiveness of their work with conversational data. To commence with, I utilized high quality SAMSum Corpus as my dataset (Gliwa et al., 2019). Various pre-processing and tokenization steps were carried out to convert the dialogue data into a compatible format for the BART model (Vaswani et al., 2017). Next, I explored and tested around with different versions of BART models in the HuggingFace Library and selected appropriate one for fine-tuning (BART, n.d.). There are large amounts of open-source pre-trained models available in this library that are particularly useful for fine-tuning. They provide a robust foundation, requiring only minor adjustments for specific tasks. The implemented solution was continuously refined and tested with different sets of hyperparameters to improve its generalization ability on unseen data. Finally, the fine-tuned model was evaluated and tested with both quantitative and qualitative metrics (Lin, 2004). To ensure that this project achieve the goals, I proposed the following objectives.

1.3 Objectives

This project focused on enhancing dialogue summarization techniques through the fine-tuning of BART, which aimed to:

- Extend BART’s capabilities to handle dialogue datasets beyond the scope of its pre-trained news or journal datasets
- Produce abstractive summaries that are coherent and retain essential information from the texts
- Enhance dialogue summarization performance by tuning the model’s hyper-parameters
- Provide a framework for summarizing conversational content from different domains in the future

To achieve these objectives, I fine-tuned the pre-trained Transformer model, BART, specifically for abstractive text summarization using the dialogues dataset. The details of this approach will be discussed in the next chapters.

2 Literature Review

Traditional summarization techniques are categorized into extractive and abstractive methods. Saxena & El-Haj (2023) highlighted that **extractive summarization** involves selecting key sentences or phrases directly from the dialogue. Several rule-based and statistical approaches have been developed over the years, with the very first reference dated back to 1958 using sentence scoring (Luhn, 1958). This includes three approaches in general namely (i) **Word Scoring**, which involves assigning scores to the most frequent words e.g. **TF-IDF**; (ii) **Sentence scoring**, which evaluates sentence features namely **sentence position** or **centrality**; and (iii) **Graph scoring**, which examines the relationships between sentences e.g. **aggregate similarity** or **TextRank** (Ferreira et al., 2013). Later, machine learning methods become prominent which depend on algorithms to learn patterns from data. For example, **unsupervised learning with skip-thought vectors** and **k-mean clustering** are used to extract central sentences in the clusters (Kiros et al., 2015; Akter et al., 2017). When NLP evolves, **abstractive summarization** has shown significant improvement in summarization performance.

Cohan et al. (2018) suggested that **abstractive summarization** generates novel sentences that capture the key context of the conversation. **Traditional Seq2Seq models** have played a pivotal role in establishing a foundation for sequence-to-sequence learning tasks (Sutskever et al., 2014). Sutskever et al. (2014) describe them as neural network architectures designed to process sequential data as input and produce a sequence of elements as output. These models use **RNNs** and later enhancements like **Long Short-Term Memory (LSTM)** and **Gated Recurrent Unit (GRU)** units to capture temporal dependencies, which is crucial for tasks beyond summarization, such as machine translation and question-answering (Cho et al., 2014; Lipton et al., 2015). Furthermore, attention mechanisms was introduced by Bahdanau et al. (2015), enabling Seq2Seq models to focus on relevant parts of the input sequence during generation and significantly improve performance. In the next two years, self-attention mechanism has completely revolutionized the field of NLP.

In recent years, **Transformers** were introduced in the paper *"Attention is All You Need"* by Vaswani et al. (2017). Unlike traditional Seq2Seq models that rely on recurrent architectures, Transformers utilize self-attention mechanisms to capture relationships between words across the entire sequence (Vaswani et al., 2017). This makes Transformers particularly powerful for both extractive and abstractive summarization. For instance, Transformer-based models like **BERT (Bidirectional Encoder Representations from Transformers)** have shown great success in extractive summarization by analysing content bidirectionally (Devlin, 2019). **BERTSum**, a variation of BERT is designed specifically for summarization (Liu & Lapata, 2019). Furthermore, models like **GPT (Generative Pre-trained Transformer)** and **BART** are widely used for abstractive summarization. They make use of autoregressive decoding methods to generate new sentences that capture content from original text (Lewis et al., 2019; Floridi et al., 2020). BART, however, combines both the benefits of bidirectional encoding and autoregressive decoding for more effective performance in generating human-like summaries (Lewis et al., 2019). More advancements like **T5 (Text-to-Text Transfer Transformer)** and **Pegasus** were introduced in the same year. T5 reframes NLP tasks as a text-to-text problem while Pegasus uses gap-sentence generation to understand document structures and longer texts (Raffel et al., 2020; Zhang et al., 2020). There are more large language models available in the Hugging Face Library for reference.

Recent studies have explored that fine-tuning enhances Transformer's performance on task-specific applications (Han et al., 2021). It involves training the model with a targeted high quality dataset tailored to the particular task (Han et al., 2021). For instance, **Llama-2-Chat**, a fine-tuned variant of Llama 2 specifically optimized for dialogue-based applications with model sizes reaching up to 70 billion parameters (Touvron et al., 2023). Despite the above advancement from Meta, there is limited studies on multi-speaker dialogue summarization. Inspired by this, I believe fine-tuning a robust pre-trained model for dialogue summarization is a promising approach in the project. Dialogues, often containing interruptions, fragmented sentences, and informal language, can be considered "noisy" in a similar sense. While models like GPT-2, BERT, and others have demonstrated success in text summarization tasks, they struggle with handling noisy data. For example, BERT only exhibits a bidirectional encoder and is not inherently designed for generating text. Based on our task's nature, the availability of labeled data, and computational resources, I believe that BART's denoising autoencoder design is well-suited for sequence-to-sequence tasks such as summarization (Lewis et al., 2020). It can effectively capture the relationship within natural language from noisy input data (Lewis et al., 2020). BART has not been extensively applied to many dialogue summarization which presents an opportunity to explore its potential in this area.

3 Dataset

In the following section, I elaborated on the dataset's characteristics by introducing its background and doing exploratory data analysis. Additionally, I discussed the preprocessing methods I employed to clean and prepare the data for training. This included tokenization, handling of special characters, and normalization steps to ensure the dataset is in the optimal format for model input. By thoroughly preparing the dataset, we could enhance the performance and accuracy of the BART model in generating high-quality dialogue summaries.

3.1 Overview

Regarding the dataset, I utilized the SAMSum corpus, a publicly available dataset created by Samsung's R&D team (Gliwa et al., 2019). This corpus contains a substantial collection of dialogues, featuring over 16,000 conversational messages and human-annotated summaries in English (Gliwa et al., 2019). It was an ideal dataset for training and fine-tuning the BART model specifically for dialogue summarization.

The SAMSum dataset, designed by linguists, mimiced real-life messenger conversations involving two or more individuals. It encompassed chit-chat, political debates, scheduling meetings, and more (Gliwa et al., 2019). Consequently, it contained a variety of text types, including formal, semi-formal, informal language, and typos. The dialogues include slang, abbreviations (e.g., "u" for "you"), emojis (e.g., 😊), and informal punctuation like ellipses ("...") etc. This reflects the spontaneous nature of real-life messaging. Besides, typos and misspellings are also common which enhance the authenticity of the conversations. Each utterance begins with the speaker's name and around 75% of the conversations are between two participants, while the remaining involve three or more individuals (Gliwa et al., 2019). Language experts then annotated them with summaries. According to Gliwa et al. (2019), this dataset did not include any confidential information or excerpts from other related datasets. I download the complete dataset from Kaggle, which included three CSV files for training, testing, and validation (Malode, n.d.). A dialogue example is illustrated in Figure 2.

```
{
 'id': '13828064',
 'summary': 'Laura will pick up Kim from work around 7, and they will come back home together.', 'dialogue':
"Laura: ok , I'm done for today-) \r\nLaura: let me know once u're free and we come back home together \r\nKim:
```

```
hmm.. 7? \r\nLaura: ok \r\nKim: cool, wait for me at work, I'll call once I get here”
 }
```

Figure 2. SAMSum dataset snapshot

3.2 Exploratory Data Analysis

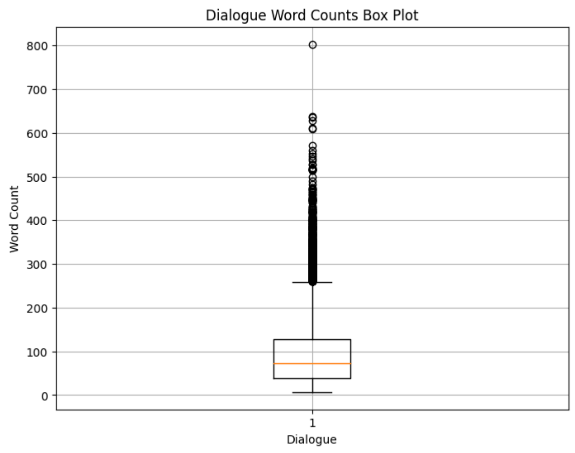
To understand the dataset better and uncover patterns in the dialogues, I carried out exploratory data analysis (EDA) in this section. The analysis was performed on word frequency, distribution of words, n-grams etc. The following visualizations helped to identify common and rare words used, word distribution patterms and the presence of special words. These provided insights into the language and content of the conversations which was crucial for understanding the characteristics of the dataset and for informing subsequent preprocessing and model training steps.

3.2.1 Text Length Distribution

To analyze the distribution of dialogues’ and summaries’ lengths in the dataset, I utilized both box plots and histograms. The box plot in figure 3 reveals that the mean dialogue length was approximately 75 words while that of summary length was less than 20 words. Notably, there are several outliers with dialogue lengths exceeding 250 words. Figure 4 further highlighted a prominent peak of 5,000 dialogues having around 0-50 words. This suggests that while shorter dialogues were more common, there were still some number of longer dialogues that extend beyond the typical range. The variability in dialogue length results in more outliers and reflects diverse conversational contexts or content complexity.

On the other hand, Figure 5 indicated that the average summary consists of approximately 18 words, with the upper and lower quartiles range being relatively narrower compared to that of dialogues. Unlike the dialogues, the human-annotated summaries exhibited fewer outliers which reflects a more consistent summarization length. Figure 6 also supported this observation, showing a peak in the 10-13 word bin with more than 2,000 summaries. Counts remained relatively stable and concentrated across the range from 8 to 22 words.

In contrast to dialogue lengths, which exhibited wide variation with many outliers and a broader distribution, summary lengths were more uniform and concise. This was evident from Figure 7 where the distribution of counts across bins (represented by the blue bars) was more even compared to the dialogue lengths (represented by the orange bars). Instead of clustering around narrow range with high counts, the summary lengths displayed a flatter distribution with similar counts across most bins. The overall shape was more spread out, indicating the uniformity in summary lengths.



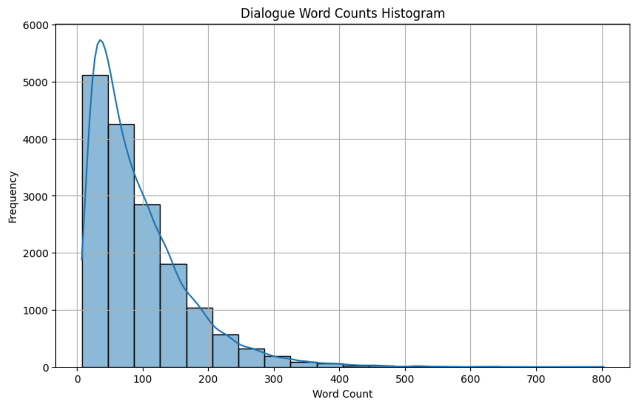
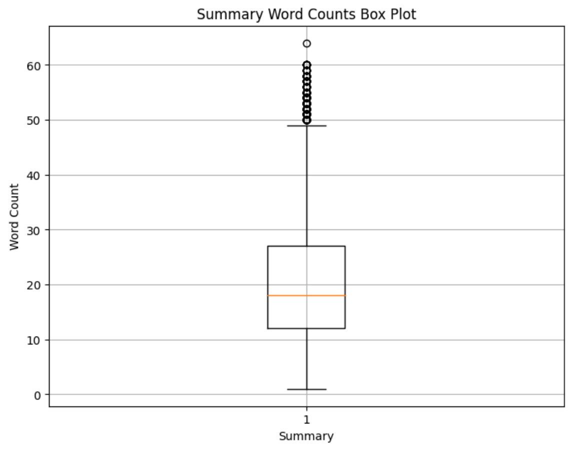


Figure 3. BoxPlot of dialogue length distribution

Figure 4. Histogram of dialogue length distribution



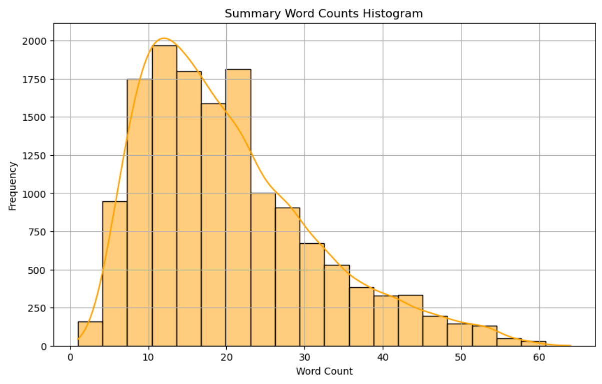


Figure 5. BoxPlot of summary length distribution

Figure 6. Histogram of summary length distribution

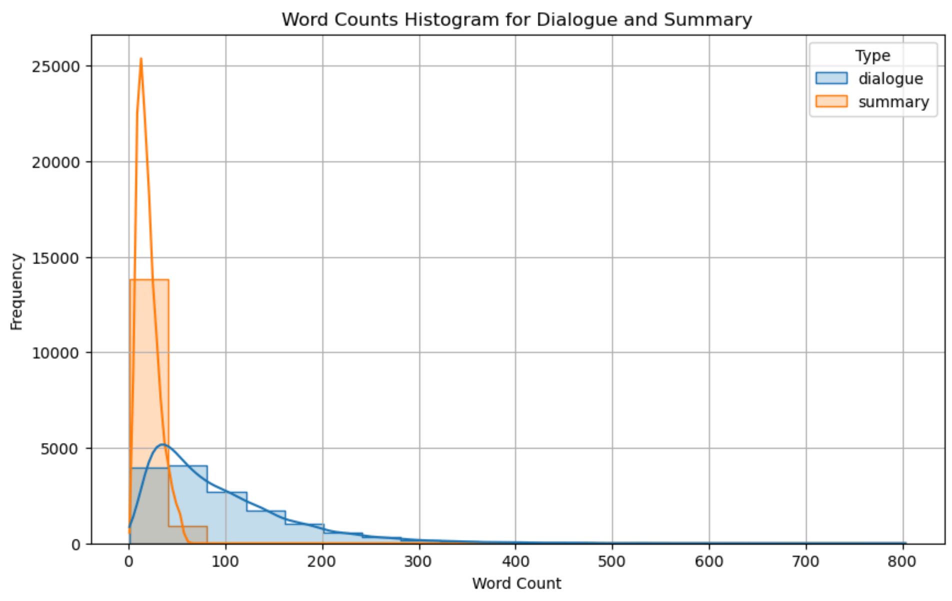


Figure 7. Histogram of word count comparison between dialogues and summaries

3.2.2 Word Frequency

By examining the word frequency bar charts in Figure 8 and 9, I noticed a Zipf's law pattern: a small number of words appear very frequently, while the majority are less common (Piantadosi, 2014). Stop words like ‘the’, ‘to’, ‘and’ were particularly prevalent and dominated the distribution in both the ‘diagoue’ and ‘summary’ columns, potentially skewing the results. The word 'i' in the 'dialogue' column appeared nearly 70,000 times, which was about seven times more frequent than the 60th most common word. To do additional analyses to complement the word frequency histogram, I examined word co-occurrence patterns to gain deeper insights into the content of the dialogues and summaries in Section 3.2.3. Moreover, I would further discuss the impacts of high occurrence of stops words and the necessity for preprocessing in Section 3.3.

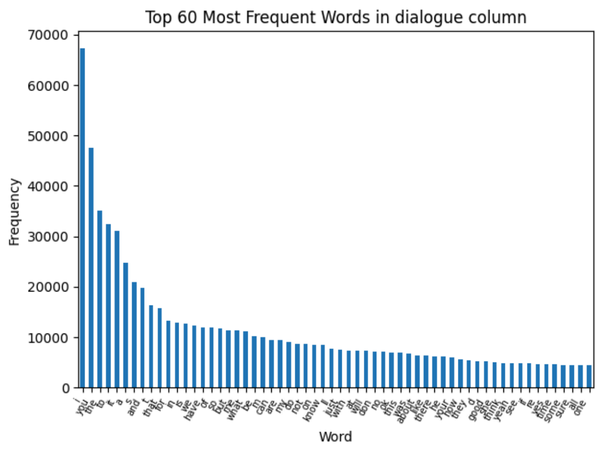


Figure 8. Top 60 most frequent words in dialogues

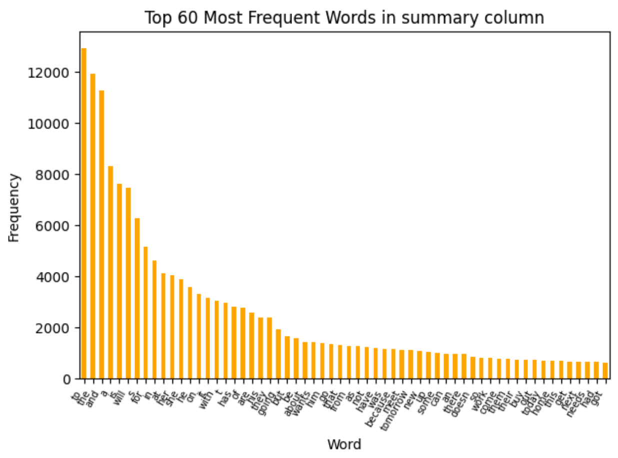


Figure 9. Top 60 most frequent words in summaries

3.2.3 N-grams Analysis

In this section, I analysed the relationship and structure between frequently occurring words and phrases by counting the distribution of n-grams and computing their correlations. N-grams are the adjacent words in a sentence with length, n (Jurafsk & Martin, 2023).

The unigram correlation matrix of dialogues, as shown in Figure 10, highlighted strong positive correlations, such as between "know" and “don" (0.37), "think" and "don" (0.28) and ‘know’ and ‘just’ (0.26). "Just" is also correlated with "really" (0.18) and "know" (0.26) which suggest common conversational patterns or expressions. For bigrams in Figure 11, the correlations were weaker, with most of the values close to 0. Notably, the bigram "don think" correlated positively with "don know" (0.046) and "don want" (0.091). The presence of "don" likely stemmed from the contraction of "don't" being split into two sub-tokens: "don" and "t".

In summary column, the unigram correlation matrix of summaries in Figure 12 were generally low. This may stem from lower repetition in summary context. Unigrams like "meet" and "pm" (0.12) showed moderate association, while bigrams in Figure 13 such as "going meet" and "15 minutes" (0.027) occurred in similar contexts. These patterns revealed how summaries were being more concise in capturing key points while dialogues showed varied, informal language use. The above analysis discovered patterns of word usage and relationships.

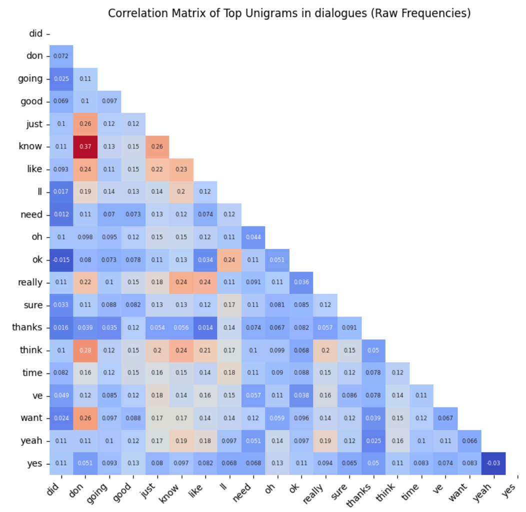


Figure 10. Heatmap of top unigrams in dialogues

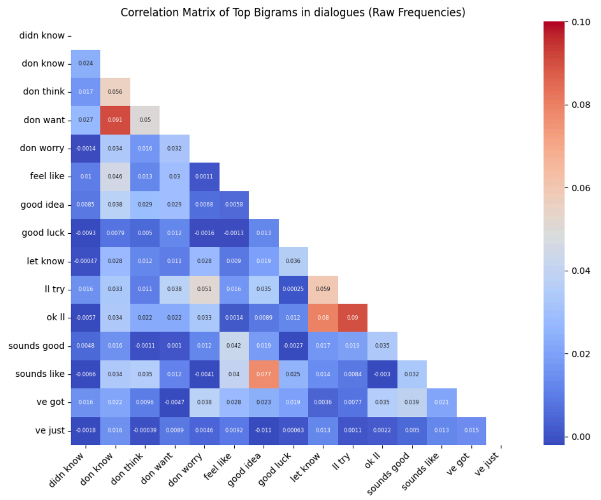


Figure 11. Heatmap of top bigrams in dialogues

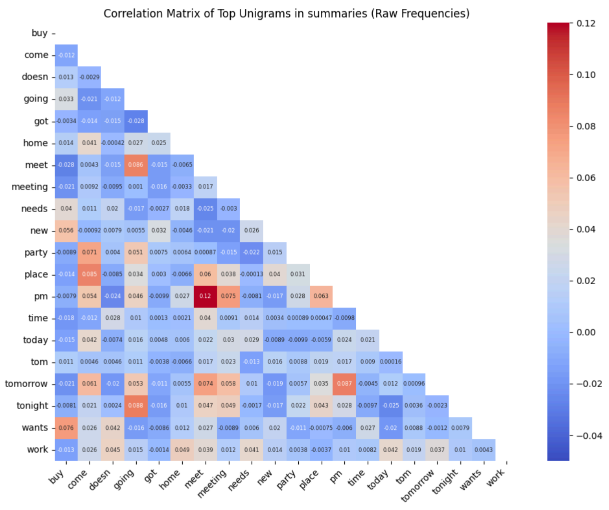


Figure 12. Heatmap of top unigrams in summaries

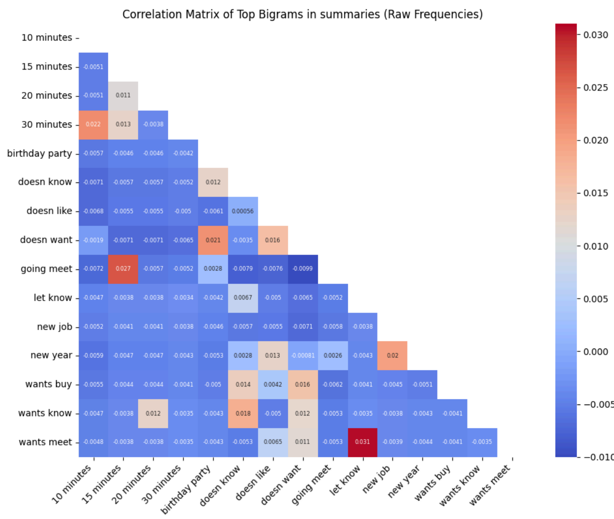


Figure 13. Heatmap of top bigrams in summaries

3.3 Data Pre-processing

In the pre-processing steps, I filtered out the HTML tags, including files and images to ensure a cleaner and more accurate text input for summarization. This process removed noise and interference from non-text elements, enhancing the quality of the data. Furthermore, I converted the pandas DataFrames into a format suitable for Hugging Face datasets. This facilitated a more compatible integration with Hugging Face Transformers for further processing.

Concerning the handling of stop words, they were retained as Transformer models, like BART, required the full context of the dialogue input to generate accurate embeddings. Besides, BART did not consider word frequency as a criterion for summarization. Instead, it focused on understanding the context of the entire input sequence. Therefore, high-frequency stop words were not problematic. Preserving stop words helped preserve context and sentence structure, which was essential for generating coherent and accurate summaries. Since the SAMSUM dataset was manually curated by Samsung's linguists and language experts, it did not consist of any missing or duplicated data, thereby providing a reliable and good-quality resource for text summarization.

3.4 Tokenization

Tokenization is vital in assisting machines to understand and consolidate the semantics of human languages by splitting them down into pieces. In this process, a matrix of input data with *N* x *d* dimensions was created where *N* was the number of tokens and *d* was the dimensionality of the embedding vectors. Tokenization methods differ depending on the granularity in breaking down the text and the specific needs of the task. It generally involved dividing the text into words, subwords, characters, or even smaller units (Vaswani et al., 2017). Each token was then mapped to a unique numerical ID. I imported `BartTokenizer` from the Hugging Face `Transformers` library to tokenize the input that was compatible with the BART language model. The final high-dimensional vector, called embeddings, was derived from the embedding layer of the BART model. The layer acted as a look-up table to return the embedding vector for each token's numerical ID (Vaswani et al., 2017).

4 Project Preliminary

This section delved into the project preliminaries in detail, covering various summarization approaches, including both extractive and abstractive methods, the different applicable model architectures such as RNNs, LSTMs, and Transformers, and the underlying mathematical theories behind text summarization. Additionally, it explored their applications in real-world scenarios and the challenges of developing a effective summarization model.

4.1 Extractive and Abstractive Text Summarization

4.1.1 Extractive Summarization

Extractive summarization selects and compiles the most relevant sentences from the original text to create a summary (Saxena & El-Haj, 2023). Unlike abstractive summarization, it directly picks sentences from the source without rephrasing and reorganizing content. It preserves the original phrasing and sentence structure. This approach uses statistical measures to assess the importance of text components, namely word frequency, sentence position etc. (Saxena & El-Haj, 2023). For instance, frequency-based methods like TF-IDF (Term Frequency-Inverse Document Frequency) is used to identify important sentences (Saxena & El-Haj, 2023). This generates summaries that accurately reflect the original content without introducing new interpretations.

4.1.2 Abstractive Summarization

On the contrary, abstractive summarization creates new sentences that capture the core ideas of the input text by paraphrasing and synthesizing information (Cohan et al., 2018). This approach generates summaries that have different wording and structure from the original text while retaining the essential meaning. Deep learning models, like RNNs, LSTMs, and Transformers, are commonly used for abstractive summarization. Models such as BERT, GPT, and T5 have achieved remarkable results through extensive pre-training on large and diverse text datasets (Islam et al., 2023). These models typically employ a sequence-to-sequence framework and self-attention mechanism that processes input sequences and predicts the corresponding output sequences (Sutskever et al., 2014).

4.1.3 Opportunities and Challenges

In this project, the abstractive summarization approach is more applicable. It excels at generating more coherent and concise summaries that effectively capture the semantic meaning of the input text (Suleiman & Awajan, 2020). Unlike extractive summarization, it performs a comprehensive analysis of the structure and semantics of the data to resemble human-generated summaries. However, the advanced capability comes with a trade-off. This approach is more resource-intensive as it requires significantly more computational resources to analyze and synthesize the input text. To conduct abstractive summarization using BART in this project, GPU was employed to achieve a more efficient performance.

4.2 Sequence-to-Sequence Architecture

4.2.1 Basic Seq2Seq Architecture

Seq2Seq models process sequential input and produce output as sequences (Bahdanau et al., 2014). In text summarization, a many-to-many sequence modeling approach is used, where the model handles multiple input tokens to produce multiple output tokens. This approach maps input and output sequences of various lengths with high flexibility (Bahdanau et al., 2014). The model can then efficiently grasp the structure and key information from the input text to generate concise summaries (Bahdanau et al., 2014).

Seq2Seq models generally consist of an encoder and a decoder, both of which are built using RNNs (Cho et al., 2014). The encoder is responsible for processing the input sequence and compressing its contextual information into a fixed-length context vector or hidden state (Bahdanau et al., 2014). This context vector is then passed to the decoder, which generates the output sequence one element at a time based on the information captured by the encoder (Bahdanau et al., 2014). Having introduced the general concept of Seq2Seq models and the encoder-decoder framework, it is essential to understand the role of RNNs within these models in the following section.

4.2.2 Recurrent Neural Networks (RNNs) in Seq2Seq

As the primary building blocks of both the encoder and decoder, RNNs enable the model to process sequential data and capture temporal dependencies effectively in the Seq2Seq model (Cho et al., 2014). While convolutional neural networks (CNNs) are tailored for grid-based data like images, RNNs are connectionist models that process sequential data one element at a time (Lipton et al., 2015). RNNs differ from standard feed-forward networks since they maintain a state that can capture information from an arbitrarily long context window (Lipton et al., 2015). This allows RNNs to map the entire sequence of previous inputs to each output due to its memory retention ability of past inputs in the hidden states (Graves, 2012). According to Hammer (2000), an RNN with enough hidden units can accurately approximate any measurable sequence-to-sequence mapping. Figure 14 shows an unfolded recurrent network (Zargar, 2021).

In Formula (1), the hidden states h_t at time t, receive information captured from input vector x_t and the hidden state h_{t-1} from the previous time step. The input x_{t-1} at time *t-1* can affect the output $\hat{y}_t$ at time *t* and beyond through the recurrent connections, as shown by Formula (2) (Lipton et al., 2015). W_x and W_y represent the weights connecting the input to the hidden layer and hidden layer to the output layer respectively, while W_h denotes the recurrent weights that link the hidden layer to itself at consecutive time steps. b_h and b_y are bias parameters. σ denotes a sigmoid activation function that introduces non-linearity into the model. Activation functions like hyperbolic tangent (tanh), ReLU are also commonly used,as shown in Formula (3) to (5).

$$h_t = \sigma(W_x x_t + W_h h_{t-1} + b_h)$$

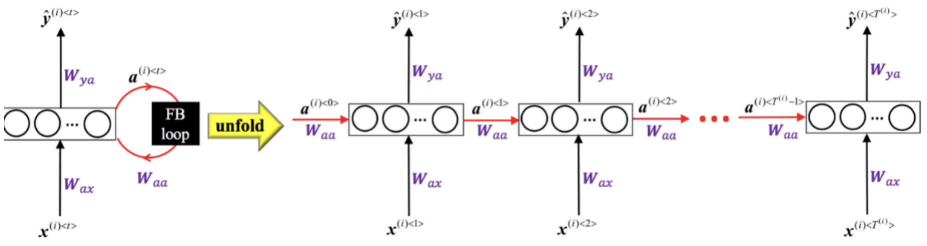
Formula (1)

$$\hat{y}_t = softmax(W_y h_t + b_y)$$

Formula (2)

$$\text{Sigmoid function: } \frac{1}{1+e^{-x}}\text{; tanh function: } \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}\text{; ReLU: } max(0, x)$$

Formula (3)-(5)



→ Training/test example: $\left(x^{(i)}, y^{(i)}\right) = \left(\left[x^{(i)<1>}, x^{(i)<2>}, \cdots, x^{(i)<t>}, \cdots, x^{(i)<T^{(i)}>}\right], \left[y^{(i)<1>}, y^{(i)<2>}, \cdots, y^{(i)<t>}, \cdots, y^{(i)<T^{(i)}>}\right]\right)$

- $x^{(i)<t>}$ = Input data point (scalar/real valued vector) at time step *t* for the *i*th training/test example
- $y^{(i)<t>}$ = Target (scalar/real valued vector) at time step *t* for the *i*th training/test example
- $\hat{y}^{(i)<t>}$ = Predicted output (scalar/real valued vector) at time step *t* for the *i*th training/test example
- $a^{(i)<t>}$ = Hidden state (real valued vector) at time step *t* for the *i*th training/test example
- W_{ax}, W_{ya}, W_{aa} = Weight matrix associated with the input, output, and hidden states respectively

Figure 14. Unfolded RNN structure

In the Seq2Seq architecture, RNNs play an important role in both the encoder and decoder components. Below shows how they function within each part:

- **Encoder:**

- **Input Processing:** The encoder processes the input sequence $x = (x_1, x_2, ..., x_T)$ using an RNN. As the RNN iterates through each token in the sequence, it generates a sequence of hidden states $h^e = (h^e_1, h^e_2, ..., h^e_T)$. Each hidden state h^e_t captures the contextual information of the input up to the current time step *t* (Bahdanau et al., 2014). In Formula (6), f^e could be a non-linear function namely LSTMs or GRU etc.

$$h^e_t = f^e(x_t, h^e_{t-1})$$

Formula (6)

- **Contextual Representation:** The final hidden state h^e_T is often used as the context vector c. This vector compresses the information of the entire input sequence and serves as a summary to be passed to the decoder (Bahdanau et al., 2014).

- **Decoder:**
  - **Sequence Generation:** The decoder is another RNN that begins with an initial hidden state, which is initialized with the final hidden state of the encoder (Bahdanau et al., 2014).
  - **Output Production:** The decoder processes the output sequence $y = (y_1, y_2, ..., y_T)$ one token at a time. At each time step, it updates its hidden state h^d_t, which captures both the information from all previously generated tokens and context vector c, as illustrated in Formula (7) (Bahdanau et al., 2014). Based on this updated hidden state, the decoder predicts the next token in the sequence using softmax activation in Formula (8). Formula (9) shows the conditional probability resulted from the prediction outcome of y_t (Maucher, 2022). During training, the model parameters θ are optimized to maximize the log-likelihood over all time steps, as shown by Formula (10).

$$h^d_t = f^d(z_t, h^d_{t-1}, c) \text{ where } z_t = \{y_1, ..., y_{t-1}\}$$

Formula (7)

$$y_t = softmax(W_y h^d_t + b_y)$$

Formula (8)

Conditional Probability: $y_t \sim p(\, y_t | \{y_1, ..., y_{t-1}\},\ c)$

Formula (9)

Maximizing log-likelihood: $argmax_{\theta} \sum_{t=1}^{T} ln\, p(y_t | \{y_1, ..., y_{t-1}\}, x)$

Formula (10)

RNNs are commonly used for sequence-to-sequence tasks such as text summarization. They are capable of handling sequential data and inputs of varying lengths and can retain information from previous inputs to generate outputs (Lipton et al., 2015). However, RNNs have slow computational speed and vanishing gradient problems (Lipton et al., 2015). The gradients used for weight updates can become very small, hindering the network's ability to learn new weights (Lipton et al., 2015). This limitation can be tackled by LSTM.

4.2.3 Long Short-Term Memory (LSTM)

RNNs face challenges with long-term dependencies due to issues like vanishing and exploding gradients. It becomes difficult for RNNs to remember information from earlier in the sequence. LSTMs allow the model to capture and maintain long-term dependencies more effectively (Lipton et al., 2015).

In LSTM, each ordinary node in the hidden layer is replaced with a memory cell which includes a node with the self-connected recurrent edge of fixed weight (Lipton et al., 2015). This ensures that information can persist over time, mitigating the vanishing gradient problem. LSTM uses three gates: the input gate i_t , forget gate f_t, and output gate o_t , which control the flow of information by selectively retaining, updating, and output information (Yin et al., 2017). All these gates are computed by applying a sigmoid function σ to the combination of the current input x_t and the previous hidden state h_{t-1}, as shown in Formulas (11) - (16) below (Yin et al., 2017). The formulas illustrate how three gates are used to update the cell state and generate the hidden state at the current time step *t* (Yin et al., 2017).

Input gate $i_t = \sigma(x_t U^i + h_{t-1} W^i + b_i)$

Forget gate $f_t = \sigma(x_t U^f + h_{t-1} W^f + b_f)$

Output gate $o_t = \sigma(x_t U^o + h_{t-1} W^o + b_o)$

$$q_t = tanh(x_t U^q + h_{t-1} - W^q + b_q)$$

$$p_t = ft_t * p_{t-1} + i_t * q_t$$

$$h_t = o_t * tanh(p_t)$$

Formula (11)-(16)

By retaining relevant information over longer time steps, LSTM can produce more coherent and contextually accurate outputs compared to traditional RNNs. Despite its computational complexity, it is particularly well-suited for tasks involving sequences where long-term dependencies are crucial.

4.2.4 Attention Mechansim in Seq2Seq Model

In addition to the basic approach, attention mechanisms was introduced which significantly enhance the model's performance (Vaswani et al., 2017). Attention allows the decoder to dynamically focus on the most relevant parts of the input sequence, instead of treating all tokens equally (Vaswani et al., 2017).

At each decoding step *t*, the relevance between the current decoder hidden state h^d_t and each encoder hidden state h^e_k is computed and represented as an attention weight α_{tk} (Bahdanau et al., 2015). Each attention weight indicates how much importance the model places on the encoder hidden state corresponding to token *k* when generating the output at decoding step *t*. With the use of the attention weights α_{tk}, the model computes a weighted sum of all the hidden states $h = (h_1, h_2, ..., h_T)$ of the encoder, represented by context vector c_t in Formula (17) (Vaswani et al., 2017; Suleiman & Awajan, 2020). This weighted sum captures the most relevant information from the input sequence at each step (Bahdanau et al., 2014). Decoder can then generate more accurate and contextually appropriate outputs (Maucher, 2022). While technologies evolve in this digital era, new Transformer architecture has been a well-known option for NLP tasks. It focuses on a self-attention mechanism that allows each input element to attend to all other input elements simultaneously, rather than sequentially. This will be discussed further in the next section.

$$c_t = \sum_{k=1}^{Tx} \alpha_{tk} h^e_k$$

Formula (17)

4.3 Transformer Architecture

4.3.1 Key Components of Transformers

Transformer is a neural network architecture that primarily uses self-attention mechanisms but not recurrent connections (Vaswani et al., 2017). It addresses potential drawbacks in RNNs ranging from fixed-length context vector, sequential processing and long-range dependencies issues (Vaswani et al., 2017). In the next subsection, I will compare Transformer with RNNs in a more comprehensive way. Transformer revolutionizes NLP to facilitate more efficient training and superior performance on various tasks such as machine translation, text summarization, and language modeling. Below shows some of the key components in a Transformer in Figure 15:

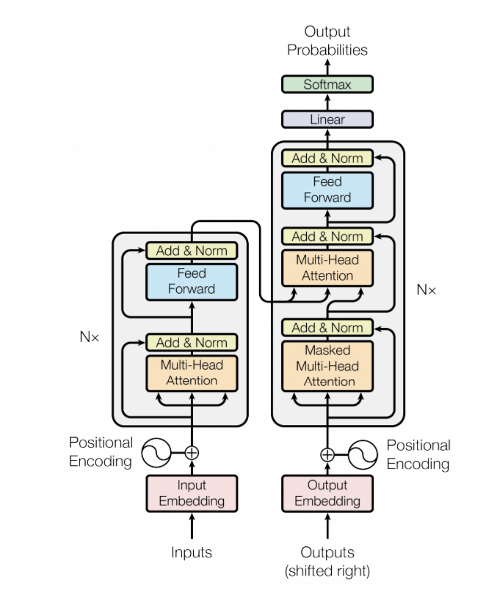


Figure 15. Transformer Architecture

1. **Self-Attention**: The self-attention mechanism determines the relevance between different tokens in the input sequence (Vaswani et al., 2017). This process attends tokens to each other and captures contextual relationships between tokens regardless of their positions (Vaswani et al., 2017). For example, when summarizing a document, the model uses this mechanism to weigh and focus on important sentences or word phrases that contribute to the overall summary. This ability ensures that important context is preserved and accurately reflected in the summary. Attention is computed below:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

where Q (Query matrix): the current token under processing, determines which parts of the input are relevant to the current token;

K (Key matrix): It determines the importance of each position in the input;

V (Value matrix): It computes the weighted sum of the input, as given by the self-attention;

d_k: the dimension of the vectors *Q, K, V;*

$softmax()$: softmax function, calculates the weights for the values

Formula (18)

2. **Multi-Head Attention**: Instead of using a single attention function, multi-head attention captures various aspects of the input text simultaneously by running self-attention in parallel (Vaswani et al., 2017). It is done by linearly projecting queries, keys and values h times into smaller dimensions, d_k, d_k and d_v respectively through learned linear projections (Vaswani et al., 2017). The attention function is then applied to each set of projected queries, keys, and values in parallel. Each attention head can then focus on different parts of the text. By combining these multiple perspectives in Formula (19), the model learns various features and generates richer representation, hence more informative summaries in text summarization tasks.

$$MultiHead(Q, K, V) = Concat(head_1, ..., headh_h)W^O$$

$$\text{where } head_i = Attention(QW_i^Q , KW_i^K, VW_i^V)$$

Formula (19)

Here, $W_i^Q \in R^{d_{model} \times d_k}$, $W_i^K \in R^{d_{model} \times d_k}$, $W_i^V \in R^{d_{model} \times d_v} v \; and \; W^O \in R^{hd_v \times d_{model}}$ are the projection matrices. They map and transform the inputs (Q, K, V) from their original dimensionality d_{model} to smaller dimensions.

3. **Positional Encoding**: Positional encoding can be used to provide position information of tokens within the sequence (Vaswani et al., 2017). Since the positional encodings and the input embeddings have the same dimension, summing them up ensures that the model understands

the sequence and order of information. It is crucial for sequential tasks like summarization to reflect the original text’s structure.

4. **Feed-forward Networks (FFN)**: A fully connected FFN is also present in each layers of the encoder and decoder. It is applied independently and uniformly across all positions in the input sequences (Vaswani et al., 2017). The FFN comprises two linear transformations with a ReLU activation in between, as shown by Formula (20) (Vaswani et al., 2017). Although these linear transformations are applied uniformly for each position, they use different parameters for each layer. Vaswani et al. (2017) mentioned that this can also be viewed as two convolutions with a kernel size of 1.

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

Formula (20)

where x is the input; W_1 and W_2 are the weight matrices for the two linear layers; b_1 *and* b_2 are the bias vectors.

5. **Layer Normalization (LN):** There is a post-layer normalization that contains an add function and a layer normalization after each attention sublayer and each FNN sublayer (Harsoor, 2023). The add function manages the residual connections and helps to retain essential information by allowing the inputs to bypass the sublayer. The layer normalization is defined as Formula (21).

$$LN(v) = \gamma \frac{v-\mu}{\sigma} + \beta$$

Formula (21)

where vector v is the input of the sublayer; μ and σ denote the mean and standard deviation of v; γ represents the scaling parameters; β is the bias.

6. **Encoder-Decoder Architecture**: The Transformer model is typically structured as an encoder-decoder architecture. The encoder processes the input sequence and generates a set of contextual representations (Vaswani et al., 2017). The decoder generates the output sequence based on the encoder’s representations, a concise summary in our case (Vaswani et al., 2017). Both of them consist of multiple layers of self-attention and feed-forward neural networks (Vaswani et al., 2017).

The input text is first passed into the embedding layer which will then output high-dimensional vectors called embeddings for each token. The embeddings are then passed into the encoder layers, followed by the decoder layers in Figure 15 (Vaswani et al., 2017). Vaswani et al. (2017) proposed that the decoder utilizes softmax activation to calculate the probabilities for each token, selecting the token with the highest probability as the predicted output. This structure provides a foundation for handling tasks like text summarization by mapping input sequences to output sequences with scalability and flexibility.

4.3.2 Comparison with RNNs

In this project, my major model was built on top of a Transformer architecture, instead of traditional RNNs. Transformer performs exceptionally well in parallel processing to handle input sequences simultaneously. Unlike RNNs which only process data sequentially and slow down training and inference, this parallelism is facilitated by the attention mechanism that allows the model to consider the entire context of the input at once (Vaswani et al., 2017). This makes Transformer significantly more efficient than traditional RNNs for training on lengthy input sequences, especially when using GPUs and TPUs for NLP tasks like summarization (Vaswani et al., 2017). Moreover, the self-attention mechanisms in a Transformer connect each position in the input sequence to every other position (Vaswani et al., 2017). Long-range relationships and dependencies can be captured easily and avoid vanishing gradient problems (Vaswani et al., 2017). Since Transformer model is capable of handling long sequential inputs without significantly increasing the training time, it is particularly beneficial for complex NLP tasks. Consequently, Transformer is preferred over traditional RNNs. I will further discuss its applications in NLP.

4.3.3 Applications of Transformers

Transformer has been widely used in various NLP tasks due to its scalability and efficiency, ranging from machine translation, text summarization to question answering. Models like BERT, GPT, T5, and BART are some of the renowned Transformer models achieving high performance on text summarization tasks (Saxena & El-Haj, 2023; Suleiman & Awajan, 2020). Since this project fine-tuned BART for dialogue summarization, I will focus on exploring its architecture in the following section. Before discussing BART, it’s essential to talk about BERT and GPT first as they provide foundational concepts and advancements in Transformer-based models that BART builds upon.

4.3.3.1 BERT and GPT

BERT is a pre-trained model focusing on understanding the context within a text by looking at both the left and right surroundings of each token of a sentence simultaneously, resulting in its bi-directional nature (Devlin, 2019). It's crucial for tasks that require deep comprehension of the input text, such as question answering or sentiment analysis. GPT is known for its ability as a high performance generative model (Floridi et al., 2020). GPT-3 was pre-trained with over 175 billion parameters by OpenAI, making it effective for generating coherent and contextually appropriate text, which is key to many creative or generative tasks (Floridi et al., 2020).

BART can be seen as a combination of the ideas from BERT and GPT. It uses a bidirectional encoder, similar to BERT, and an autoregressive decoder akin to GPT (Lewis et al., 2019). This combination allows BART to excel in tasks like text summarization, translation, and other sequence-to-sequence tasks. Therefore, I decided to leverage BART for dialogue summarization in this project.

4.3.3.2 BART: Bidirectional and Auto-Regressive Transformers

BART was introduced by Facebook AI Research in 2019. Since it is pre-trained using a denoising objective, it learns to reconstruct from a corrupted text (Lewis et al., 2019). It features a bidirectional encoder and an autoregressive decoder (Lewis et al., 2019). In this section, I will briefly examine BART's architecture and functionality.

BART is based on a Transformer framework and is trained using a denoising autoencoder objective, which predicts and reconstructs corrupted input by observing bidirectional context and minimizing reconstruction loss (Lewis et al., 2019). During pre-training, a variety of corruptions, like sentence permutation, token deletion, infilling etc. are performed (Lewis et al., 2019). BART's encoder is inspired by BERT's architecture and can fully consider the context from both directions within a sentence and understand complex language structures. To illustrate, the encoder predicts the masked tokens, B and D, by analyzing the surrounding tokens: A, C, and E in the input sentence in Figure 16 (Lewis et al., 2019). The encoded contextual representation is then passed through a series of stacked decoders to produce the final output.

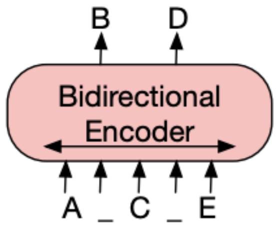


Figure 16. Bidirection Encoder illustration

The decoder of BART draws from GPT's autoregressive decoder architecture to generate subsequent words in a sequence. Figure 17 illustrates an autoregressive decoder for word generation (Lewis et al., 2019). The decoder is pre-trained on a large text corpus and undergoes supervised fine-tuning for domain-specific prediction tasks (Lewis et al., 2019). The objective is to minimize the negative log-likelihood as shown below:

$$L_{LM}(T) = \ - \frac{1}{|T|}\sum_{i=1}^{|T|} logP(t_i|t_{i-k},..., t_{i-1}\,; \theta)$$

$$L_{FT}(T) = \ - \frac{1}{|T|}\sum_{i=1}^{|T|} logP(y_i|x_1,..., x_n)$$

$$L_{BART}(C) = L_{LM}(C) + \lambda L_{FT}(C)$$

Formula (22)- (24)

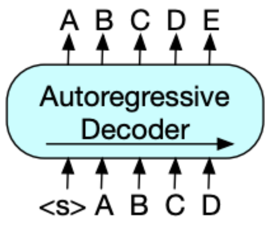


Figure 17. Autoregressive Decoder illustration

In Formula (22), $L_{LM}(T)$ represents the log-likelihood loss for a sequence T. It measures how well the model predicts the sequence of tokens. $P(t_i|t_{i-k},..., t_{i-1}\,; \theta)$ is the probability assigned to the token t_i given the preceding tokens $t_{<i}$ and model parameters θ. $L_{FT}(T)$ is the fine-tuning loss for a domain-specific dataset T. $P(y_i|x_1,..., x_n)$ represents the probability of the target label y_i given the input features $x_1,..., x_n$. The goal is to minimize the difference between the predicted probabilities and the

actual distribution. The two objectives, $L_{LM}(T)$ and $L_{FT}(T)$ are combined as the loss function of BART (Lewis et al., 2019).

Consequently, BART is a denoising autoencoder that follows the Transformer architecture and leverages bidirectional encoders and autogressive decoder (Lewis et al., 2019). In Figure 18, the corrupted sentence on the left is encoded with a bidirectional model and the likelihood of the original sentence is computed using an autogressive decoder (Lewis et al., 2019). It effectively captures dependencies in both directions while ensuring coherence in the generated text (Lewis et al., 2019; Devlin et al., 2019; Radford et al., 2018). Detail architecture of this model will be discussed in Section 6.3.

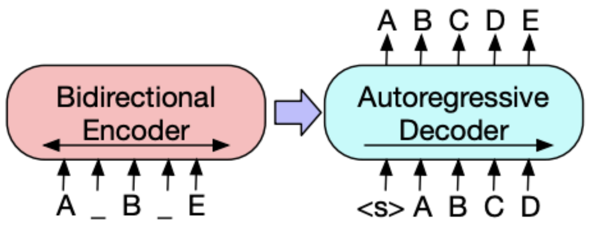


Figure 18. BART encoder-decoder illustration

5 Mini Project: Text Summarization using RNN

To get more familiarized with the overall implementation of text summarization using Seq2Seq model from scratch, I conducted the mini-project below which was a simple model constructed by RNNs.

5.1 Model Architecture

The model is an RNN-based Seq2Seq model with LSTM units. The architecture was designed to handle variable-length input and output sequences which was flexible for language processing tasks like text summarization. It is illustrated as follows:

Encoder-Decoder Structure:

I set the hidden state size to 200, balancing between the model's complexity and comuptational efficiency. This model was designed primarily for knowledge consolidation rather than for achieving optimized performance. Consequently, only a single LSTM layer was utilized in both the encoder and decoder. Below shows the structure in detail:

- **Encoder:**
    - Input Layer: Allows variable-length input to be passed into the model
    - Embedding Layer: Converts input tokens into dense vectors of 200 dimensions. Setting "mask_zero" equals True allow the model to ignore padding values
    - LSTM Layer: Processes the embedded input sequence and generates hidden states. Setting "return_seqeunces" equals True outputs the hidden states for each time step of the input sequence
- **Decoder:**
    - Input Layer: Takes target sequences of variable lengths
    - Embedding Layer: Similar to the encoder, this layer maps target tokens to dense vectors
    - LSTM Layer: Uses the encoder's final hidden and cell states as the initial states to generate output sequences
    - Dense Layer: Projects the decoder's output to the target vocabulary size, applying a softmax activation function to predict the next token

5.2 Model Implementation

Since this was a side project to understand how text summarization was built from scratch using traditional RNN, the set-up and data pre-processing differed from what was described in Sections 3.3 and 3.4 for BART implementation. The training setup for this Seq2Seq model involved several steps and choices to ensure effective learning and performance.

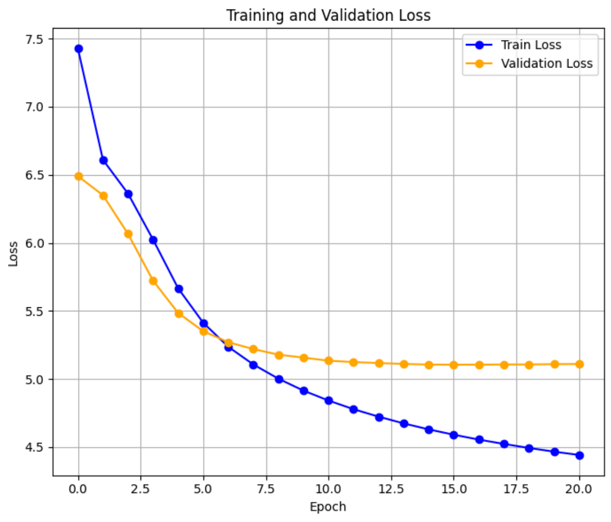
- **Data Preprocessing:** I used the `Tokenizer` class from the `Keras` library in TensorFlow to map input and target tokens into integer representations. Then, I applied padding to ensure uniform sequence lengths across all input and output sequences.
- **Loss Function and Optimizer:** The loss function used was sparse categorical cross-entropy which is suitable for target labels that are integer-encoded. Adam optimizer was also adopted, which was well-suited for handling large datasets and complex models.
- **Training Parameters:**
    - Batch size: 64
    - Epochs: 21
    - Learning rate: 0.0002, which decreased exponentially at each epoch by using a scheduler

- **Regularization Techniques:** Early stopping and learning rate scheduler were included to prevent overfitting.
- **Evaluation metrics:** Training and validation accuracy and loss were measured to monitor the model performance.

5.3 Training and Testing Results

After implementing the training and testing processes, the loss and accuracy during training and validation are plotted in Figure 19 and 20. The training loss decreased steadily and converges and the training stopped at 21 epochs. As for training accuracy, it increased above 70% while the val accuracy fluctuated. Testing accuracy has achieved 70.45%. However, I observed a slight overfitting and the training loss was still high, around 4.5. The generalization performance can also be improved by further adjusting the model architecture and regularizations.

In this mini-project, It served as a prototype for me to learn how to handle sequential data recurrently with LSTM encoder-decoder model from scratch. Future improvement work could focus on optimizing the dataset and refining the model architecture, such as incorporating attention layers.



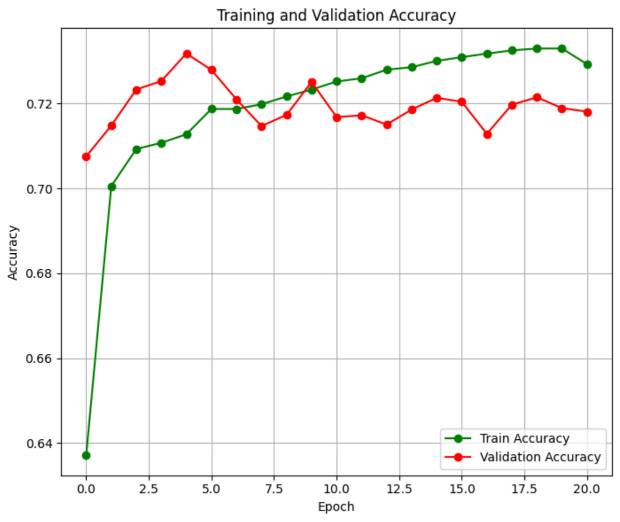


Figure 19. RNN model's Training and Validation loss

Figure 20. RNN model's Training and Validation Accuracy

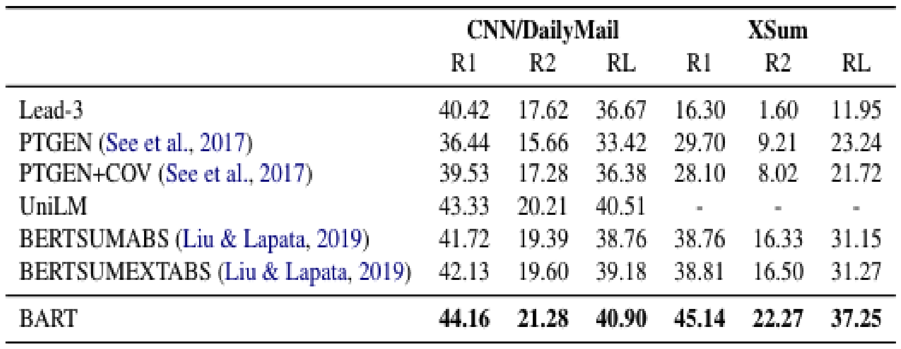
6 Text Summarization using BART

In the previous section, I explored the implementation of a mini project using RNNs, specifically LSTMs for text summarization task. While LSTMs provide a baseline for sequence-to-sequence modelling, they have certain limitations, particularly in handling complex and lengthy text sequences effectively (Lipton et al., 2015). On the contrary, BART is a Transformer-based model that combines bidirectional encoding with autoregressive decoding, making it excel at text generation tasks like summarization (Lewis et al., 2019). In the following sections, I provide details on the fine-tuning process of BART model for dialogue summarization.

6.1 Overview

According to section 4.3.3.2, it highlights the advantages of BART as a pretrained model with a denoising objective and its use of self-attention in a Transformer architecture (Lewis et al., 2019). Fine-tuning BART with a dialogue-specific dataset significantly enhances the quality of generated summaries in complex dialogue summarization by adapting the model to understand the unique structure and language style of dialogues (Han et al., 2021). This transfer learning process involves supervised training on small, high-quality labeled data and improve BART’s ability to generate accurate, contextually relevant summaries (Han et al., 2021).

Morover, BART allows developers to run it locally on private servers and GPUs which eliminates the need to upload documents to cloud providers like OpenAI, as required with GPT-3. This facilitates greater data privacy and control over the infrastructure. Developed by Facebook (Meta) AI, BART is open-source and freely available through Hugging Face's model repository, offering more flexibility than proprietary models like GPT-3.5 and GPT-4. Lewis et al. (2019) also revealed that BART surpasses all previous extractive approaches and Transformer-based models, including BERT, in summarization tasks on datasets like CNN/DailyMail and XSUM as shown in Figure 21.



	CNN/DailyMail			XSum		
	R1	R2	RL	R1	R2	RL
Lead-3	40.42	17.62	36.67	16.30	1.60	11.95
PTGEN (See et al., 2017)	36.44	15.66	33.42	29.70	9.21	23.24
PTGEN+COV (See et al., 2017)	39.53	17.28	36.38	28.10	8.02	21.72
UniLM	43.33	20.21	40.51	-	-	-
BERTSUMABS (Liu & Lapata, 2019)	41.72	19.39	38.76	38.76	16.33	31.15
BERTSUMEXTABS (Liu & Lapata, 2019)	42.13	19.60	39.18	38.81	16.50	31.27
BART	**44.16**	**21.28**	**40.90**	**45.14**	**22.27**	**37.25**

Figure 21. BART's superior performance on new article dataset

6.2 Implementation Dependencies

I built my model utilizing the Hugging Face `Transformers` library, which is renowned for its extensive collection of pre-trained models and tools for various NLP tasks (BART, n.d.). This library offers open-source and user-friendly APIs so researchers and people interested in this fields can access and fine-tune different models easily such as BERT, GPT, T5, and BART. As shown in Table 1, several versions of the BART Transformer models are available namely BART-Large, BART-Base, BART-Large-XSUM, and BART-Large-CNN. In this report, I used BART-Large-XSUM due to their larger pre-trained capacities and rich parameters (BART, n.d.). I also experimented with BART-Base which is a smaller model to compare their performance. Furthermore, the library can be integrated with deep learning frameworks like TensorFlow and PyTorch in order to facilitate straightforward implementation and fine-tuning. In particular, I loaded the tokenizer (`BartTokenizer`) and BART architecture (`BartForConditionalGeneration`). For fine-tuning, I imported `Seq2SeqTrainer`. The library is a flexible and robust community that supports the sharing and optimization of pre-trained models for different tasks.

	facebook/ bart-base	**facebook/ bart-large**	**facebook/ bart-large-cnn**	**facebook/ bart-large-xsum**
Number of parameters	139M	345M	400M	400M
Primary Use Case	General-purpose NLP tasks	General-purpose NLP tasks	Summarization	Extremely concise summarization

Architecture	6 encoder and decoder layers	12 encoder and decoder layers	12 encoder and decoder layers	12 encoder and decoder layers
Pre-trained dataset	A diverse collection of text	A diverse collection of text	Fine-tuned on CNN/Daily Mail articles	Fine-tuned on XSum dataset
Computational Complexity	Lower	Higher	Higher	Higher

Table 1. Available versions of BART

6.3 Model Architecture

I loaded the BART-LARGE-SUM model from Hugging Face `Transformer` Library for fine-tuning in the later sections. Below is a detailed breakdown of its architecture:

1. **Embedding Layers**

(a) Shared Embedding: Maps input tokens into a dense vector representation of 1024 dimensions. This embedding layer includes a vocabulary of 50,264 tokens, with padding for special tokens.

(b) Positional Embedding: Aligns token embeddings with their positions in the sequence and encodes positional information up to 1026 positions

2. **Encoder**

The encoder consists of 12 layers to process the input sequence, each equipped with:

(a) Self-Attention Mechanism: Details discussed in Section 4.3.1

(b) Feed-Forward Network: Consists of two linear transformations: expands the dimensionality from 1024 to 4096, followed by projects it back to 1024

(c) Activation Function: Uses GELU (Gaussian Error Linear Unit) for non-linearity

(d) Layer normalization: Is applied before and after the self-attention and feed-forward network to stabilize training and improve convergence

3. **Decoder**

The decoder also consists of 12 layers to generate the output sequences using the encoded representation, each equipped with:

(a) Self-Attention Mechanism: Similar to the encoder, with an additional encoder-decoder attention layer

(b) Feed-Forward Network: Similar to the encoder

(c) Layer normalization: Similar to the encoder

4. Output Layer

(a) Linear Transformation: Transform the final output by projecting the 1024-dimensional embeddings to a vocabulary size of 50,264. Logits for each token in the vocabulary are generated to predict the next token in the sequence.

Given the above architecture, the process of dialogue summarization involved several key steps. First, the input dialogue was tokenized to create a sequence of numerical IDs and then transformed into high-dimensional embedding vectors through the embedding layer. These embeddings were passed to the encoder, which utilized a self-attention mechanism, specifically multi-head attention, to process the entire input sequence. The 12-layer bidirectional encoders captured rich contextual information from both past and future tokens. The output from the encoder was a latent vector or was called a contextualized representation (Lewis et al., 2019).

Next, this latent vector was fed into the 12-layer decoder, an autoregressive mechanism that attended to past tokens and the latent vector when generating the next token in the sequence. Both encoder and decoder incorporated residual connections and layer normalization, which helped to stabilize training and speed up convergence. After processing, the decoder produced a high-dimensional vector, which was then transformed through a linear layer to create logits. A softmax function was applied to these logits to generate a probability distribution over the vocabulary for the selection of the predicted token (Lewis et al., 2019). This token was then passed back into the decoder as the previous token for the next prediction. This process continued until the end-of-sequence token, <eos> was reached. In this project, I performed the above procedures for dialogue summarization with numerous hyperparameters tuning, as demonstrated in the following section.

6.4 Fine-Tuning BART

6.4.1 Training Details

After understanding the architecture of the BART model, I performed fine-tuning which trained the pre-trained model with high quality data for dialogue-specific summarization using NVIDIA TESLA A100 GPU in the UQ's high-performing computing cluster. To perform fine-tuning process of BART-LARGE-XSUM, below are some major initial training details:

1. **Number of Epochs**: **35**
2. **Learning Rate**: **0.00001**, a commonly used base for fine-tuning pre-trained models (Sun et al., 2020)
3. **Effective Batch Size: 16**, calculated by multiplying batch size per device, set as 8 and gradient accumulation, set as 2
4. **Weight Decay**: 0.01, regularizer that prevents overfitting
5. **Evaluation Strategy**: evaluated at the end of each epoch
6. **Logging**: every 10 steps, metrics are logged after every 10 training steps
7. **Evaluation Metric**: The `compute_metrics` function calculates ROUGE, particularly useful for summarization tasks which evaluates ROUGE-1 (unigram overlap), ROUGE-2 (bigram overlap), ROUGE-L (longest common subsequence per sentence)
8. **Early Stopping:** a technique to stop training if the validation loss does not improve for 3 consecutive epochs
9. **Optimizer:** AdamW
10. **Loss Function:** Cross Entropy Loss

Throughout the fine-tuning process, I used training and validation datasets which were tokenized based on the BART tokenizer. I first set up a `Seq2SeqTrainer` that specified the model, training arguments, datasets, tokenizer, evaluation metrics and data collator that handled dynamic batching and . The data collator automated padding and batching of variable-length sequences. The model was trained for 35 epochs using the training dataset. At the end of each epoch, evaluation was performed with ROUGE metrics. Validation dataset was used since it was essential for assessing how well the model generalized to unseen data. The model's predicted summaries were compared against the ground truth labels, i.e. human annotated summaries in the validation dataset (Lin, 2004). In the evaluation phase, candidate and reference summaries were first decoded from token IDs into human-readable text using the tokenizer. The ROUGE scores were then computed using the `metric.compute` function with stemming enabled.

In Figure 22, it shows the training and validation loss during 35 epochs. While the training loss decreased and converged, the validation loss increased which indicated overfitting. This suggested that the model was unable to generalize well to new data and instead fitted the noise in the training data. To resolve this issue, I made adjustments to the hyperparameters and added more regularization techniques. These changes helped reduce the overfitting, as shown in Figure 23, where the validation loss decreased to a level slightly above the training loss. Moreover, I further experimented with the BART-Base model. The smaller and less complex nature of the BART-Base model made it less prone to overfitting, and it showed better generalization to new data, as shown in Figure 24. The following changes were made to mitigate overfitting:

- Added **Early Stopping,** model stopped at 4 epochs
- Increased **Batch Size** to **64**
- Adjust **Learning Rate** using **Cosine scheduler**, started at 0.0002 and decreased it following cosine curve
- Applied **Gradient Clipping**
- Set **Dropout** to **0.2**, which randomly dropped out 20% of the neurons in the network
- Set **Attention Dropout** to **0.1,** which randomly dropped out 10% of the attention weights in the self-attention and cross-attention layers

Each of the above hyperparameters has an irreplaceable role in the fine-tuning process. In the next sections, I investigated each hyperparameter and analyzed their importance when training the model.

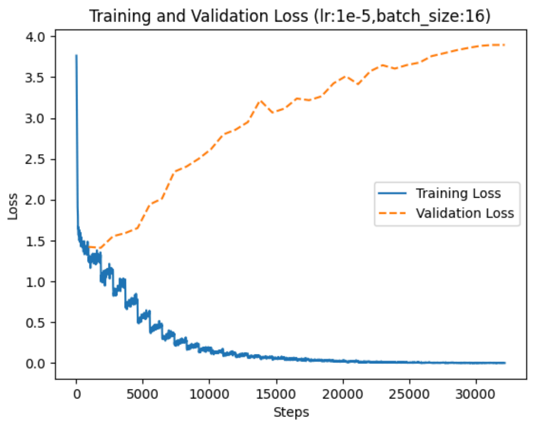


Figure 22.BART-Large-XSUM's Loss before regularization

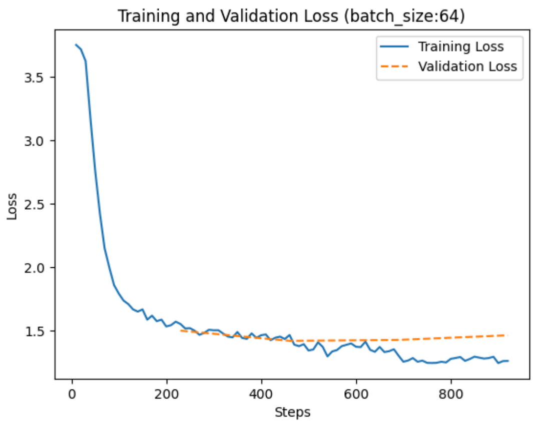


Figure 23. BART-Large-XSUM's Loss after regularization

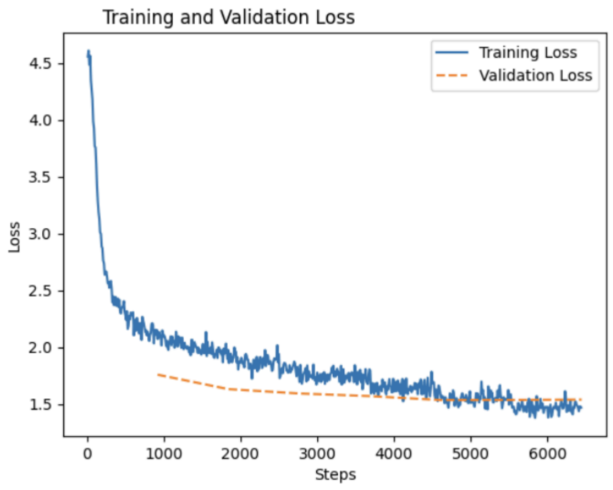


Figure 24. BART-Base's Loss after regularization

6.4.2 Adjustment of Batch Size

Batch size has played a important part in improving the model's performance by influencing the stability of gradient updates (Smith, 2018). I initially set the **batch size to 8**, which led to noisier gradients and fluctuations in the training loss curve, as shown in Figure 25. Smaller batch sizes often result in more variance during gradient updates. To tackle this, I increased the **batch size to 16** to produce a smoother training loss curve, as illustrated in Figure 26. I noticed that the increase of batch size to a reasonable level indicates more stable gradient updates and a more consistent descent in loss. Ultimately, I settled on a **batch size of 64** to achieve stable gradient updates and consistent loss descent, allowing the model to learn more effectively.

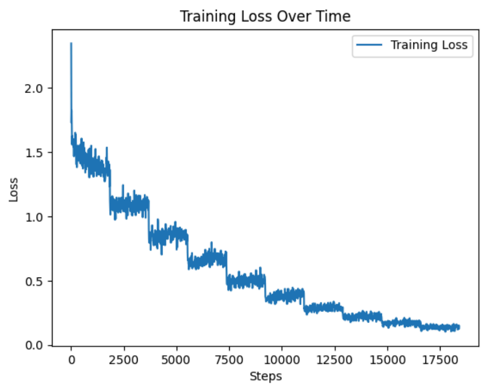


Figure 25. BART-Large-XSUM's Loss trained with batch size 8

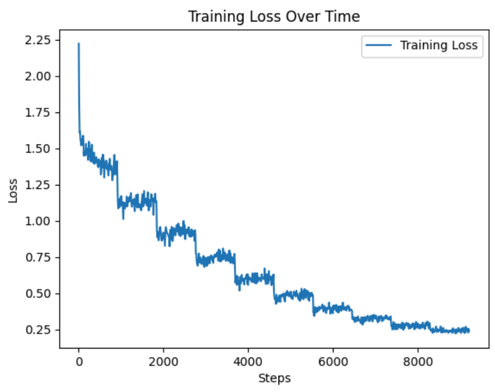


Figure 26. BART-Large-XSUM's Loss trained with batch size 16

6.4.3 Adjustment of Learning Rate

Apart from the adjustment on batch sizes, I also experimented with different learning rates during training. I have adjusted the learning rate slightly while keeping the other parameters constant. A learning rate of 2e-5 is commonly used as a base rate for fine-tuning pre-trained model (Sun et al., 2020).

With the **smaller learning rate of 1e-5** in Figure 27, the loss curve was smoother, showing more gradual and controlled training. The loss was decreasing steadily and approached a lower value, around 0.5, but there are still some minor oscillations. The model was approaching convergence but hasn't fully stabilized. In contrast, the model with the **higher learning rate of 5e-5** converged faster and the loss dropped significantly at the end of each epoch in Figure 29. There were slightly larger fluctuations during training which indicates that while the model was learning faster, it was instable. Furthermore, the significant drop in loss at the end of each epoch was due to fast learning and large updates to the weights (Smith, 2018). The model may reach a local minimum quickly, missing the global minimum instead. It also led to overfitting and poor generalization on unseen data (Smith, 2018).

Given these observations, it is clear that the choice of learning rate plays a vital role for the model's training performance. While a smaller learning rate results in stability, it is less efficient. Conversely, the higher learning rate facilitates quicker convergence but introduces instability and the risk of overfitting (Smith, 2018). To strike a balance between stability and efficiency, I implemented **a learning rate adjustment strategy using a cosine scheduler** in my final model. I started with a high learning rate of 0.0002 and applied the cosine decay technique to gradually decrease the learning rate following a cosine curve throughout training. This approach expedited convergence in the beginning and stabilize training and improve generalization by reducing the rate over time.

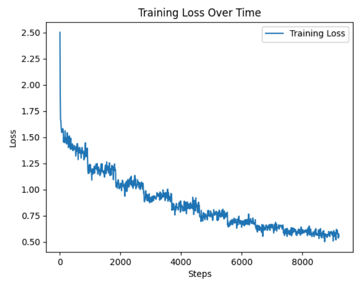


Figure 27. Learning Rate: 1e-5

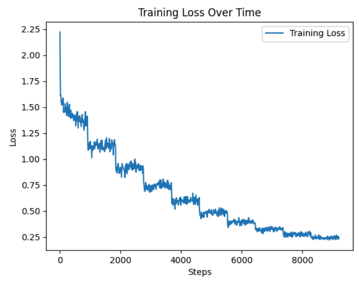


Figure 28. Learning Rate: 2e-5

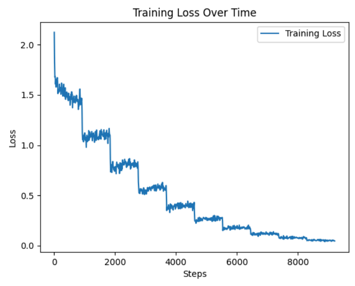


Figure 29. Learning Rate: 5e-5

6.4.4 Regularization

Besides the adjustment on batch size and learning rate, I further implemented several regularization techniques including early stopping, gradient clipping and dropout to prevent overfitting and improve model generalization.

Early stopping was used to monitor validation loss during training. The model stopped when there was no improvement after 2 epochs. This prevented overfitting to the training data while maintaining optimal performance on unseen data. Additionally, **gradient clipping** was added to stabilize training since this technique limited the magnitude of gradients during backpropagation and prevented the issue of exploding gradients in the networks. This led to smoother updates to the model parameters as well as steady convergence. Lastly, **dropout** was set to 0.2 to randomly remove 20% of neurons during training. Similarly, **attention dropout** was set to 0.1 to randomly cancel out 10% of attention weights n the self-attention and cross-attention layers. These dropout techniques prevented overfitting by prohibiting the model from over-relying on specific linguistic patterns or features (Hernández-García, 2018). This was particularly useful for large models like BART that might easily overfit on small datasets.

6.5 Evaluation using ROUGE metrics

To assess the performance of my summarization model, I primarily used the ROUGE score as the evaluation metric. I loaded the ROUGE metrics from Hugging Face's `Evaluate` library (ROUGE, n.d.). The ROUGE score is a widely recognized method for comparing the quality of generated summaries against human-annotated reference summaries (Lin, 2004). It is a automated evaluation process of counting overlapping units, such as word sequences, n-grams, and word pairs, between the generated and reference summaries (Lin, 2004). ROUGE provides a comprehensive evaluation using ROUGE-N and ROUGE-L. ROUGE-N evaluates the overlap of n-grams and ROUGE-L measures the longest common subsequence (LCS) between the candidate and reference summaries (Lin, 2004).

Formula (25) denotes the recall of ROUGE-N which is the fraction of n-grams in the reference summary that are also present in the candidate summary (Lin, 2004). $Count_{match}(gram_n)$ represents the number of matching n-grams. To compute the precision, the denominator of Formula (25) should be changed from the total number of n-grams in the reference summary to the candidate summary. Regarding the the three formulas in Figure (26)-(28), R_{LCS}, P_{LCS}, F_{LCS} are the precision, recall and F-measure of ROUGE-L (Lin, 2004). These scores range from 0 to 100, with higher values indicating a greater degree of overlap between the two summaries (Lin, 2004). The higher the score, the better the model's performance. This indicates that the model is effective at generating summaries that closely resemble human-annotated summaries.

$$\text{ROUGE-N} = \frac{\sum_{S \in \{ReferemceSummaries\}} \sum_{gram_n \in S} Count_{match}(gram_n)}{\sum_{S \in \{ReferenceSummaries\}} \sum_{gram_n \in S} Count(gram_n)}$$

Formula (25)

$$R_{LCS} = \frac{LCS(C,S)}{len(S)}$$

$$P_{LCS} = \frac{LCS(C,S)}{len(C)}$$

$$F_{LCS} = \frac{(1+\beta^2)R_{LCS}P_{LCS}}{R_{LCS}+\beta^2 P_{LCS}} \quad \text{where } \beta = \frac{R_{LCS}}{P_{LCS}}$$

Formula (26)- (28)

Table 2 shows the evaluation performance of the fine-tuning BART-LARGE-XSUM model. I evaluated the model using ROUGE Score on the validation set. The best evaluation ROUGE score was reached in just 4 epochs and it took around 30 minutes. The findings reveals some insights into the model’s performance over time:

1. **ROUGE-1 Scores:** The score at the final epoch has a high recall of 59% and all measures exceeds 50% in general. This indicates that the model consistently captured a good level of keywords relevant to the summaries(Walker II, n.d.).
2. **ROUGE-2 Scores:** The ROUGE-2 score has a relatively low recall of 33% which suggests moderate performance and the model's fair ability in capturing complex relationships in conversation (Walker II, n.d.).
3. **ROUGE-L:** These scores reflect the model's ability to capture the longest common subsequence, generally around 50%. Since score above 40% is regarded as good performance, this suggests that the model was effectively maintaining the overall structure and retains significant elements of the source text (Lin, 2004; Walker II, n.d.).

Overall, the model had a promising performance across ROUGE metrics, with particularly strong ROUGE-1 and ROUGE-L scores. However, the relatively low ROUGE-2 scores suggests that the model may need additional tuning to improve the capture of bigram relationships. In Table 3, the performance across epochs was evaluated, revealing a gradual increase in the F1 score, which stabilized at the 4th epoch due to the absence of significant changes in the ROUGE score. While these ROUGE scores provide quantitative insights, qualitative human evaluations are also essential to ensure the generated summaries meet semantic and syntactic quality standards. Therefore, both quantitative and qualitative analysis were incorporated in testing.

	ROUGE 1	**ROUGE 2**	**ROUGE L**
Precision	51.11	27.92	43.26
Recall	59.94	32.95	50.47
F-measure	52.43	28.58	44.22

Table 2. Evaluation performance of the last epoch

Epoch	**ROUGE 1**	**ROUGE 2**	**ROUGE L**
1	49.45	25.25	41.04
2	51.70	28.07	43.32

3	52.32	28.29	43.75
4	52.43	28.58	44.22

Table 3. F1 Scores across epochs

6.6 Test Results and Analysis

In this section, I tested my model with the test dataset by summarizing each dialogue and analyse the results quantitative and qualitatively. While the ROUGE scores provided a quantitative measure of how well a generated summary overlaps with a reference summary in terms of n-grams, it did not capture the quality of summary in full (Lin, 2004). For instance, ROUGE focused on surface-level matching, but it did not evaluate the synthetic quality namely coherence, readability, logics and fluency of the summary. Therefore, qualitative analysis was needed to assess whether the summary is not only accurate but also meaningful and well-structured. The tables in the below sections compares the summary generated by vanilla BART and the fine-tuned model as well as the ground truth. Quantitative measures was also included using ROUGE scores to show comprehensive result analysis. As for efficiency, the generation of each summary took around 10-15 seconds on average using GPU.

6.6.1 Test Example 1

As discussed in Section 1.1, summary of Input Dialogue 1 from vanilla BART fell short of capturing accurate information and introduced false new context. In Table 5, after fine-tuning with a dataset of 1,000 training examples, the model was able to capture the major content of the conversation. However, it still struggled with correctly identifying the subjects involved: It was Amanda who suggested asking Larry, not Hannah. Further training with a full dataset was performed afterward to enhance overall accuracy in the relationship and context, as shown in the fine-tuned model's summary. The final summary demonstrated a significant improvement in both content and accuracy. For instance, it correctly identified the relationships between Amanda, Hannah, Larry, and Betty without introducing false information.

ROUGE 1	ROUGE 2	ROUGE L
53.66	20.51	39.02

Table 4. ROUGE performance of Test Example 1

Input Dialogue 1	
Hannah: Hey, do you have Betty's number? Amanda: Lemme check Hannah: <file_gif> Amanda: Sorry, can't find it. Amanda: Ask Larry Amanda: He called her last time we were at the park together Hannah: I don't know him well Hannah: <file_gif> Amanda: Don't be shy, he's very nice Hannah: If you say so.. Hannah: I'd rather you texted him Amanda: Just text him 🙂 Hannah: Urgh.. Alright Hannah: Bye Amanda: Bye bye	
Reference Summary (Ground Truth)	
Hannah needs Betty's number but Amanda doesn't have it. She needs to contact Larry.	
Pre-trained Model	**Fine-tuned Model** (using 1000 rows of data)
A few days ago, I was talking to my friend Hannah, and we were looking for her friend Betty.	Amanda can't find Betty's phone number. Hannah suggests asking Larry for Betty's number.
Fine-tuned Model	
Hannah is looking for Betty's number. Larry called Betty the last time Amanda and Hannah were at the park. Hannah doesn't know him very well, so Amanda suggests she texts him.	

Table 5. Summaries generated for Test Example 1

6.6.2 Test Example 2

I further experimented with more examples. The pre-trained model's summary of Dialogue 2 in Table 7 introduced an incorrect implication that Rita and Tina worked at the same job, which was not explicitly stated in the dialogue. In contrast, the fine-tuned model's summary was more concise and accurate with clear emphasis on the shared sentiment of boredom in one single precise sentence, without incorrect assumptions. However, the evaluation scores in Table 6 did not reflect this since scores are relatively low due to the mismatch of n-gram between the candidate and reference summaries. The reference summary included the detail that four hours remained for work, which may not have been the primary focus for me. This highlighted the importance of balancing between human evaluation and quantitative measures in NLP.

ROUGE 1	ROUGE 2	ROUGE L
27.27	20.00	27.27

Table 6. ROUGE performance of Test Example 2

Input Dialogue 2	
Rita: I'm so bloody tired. Falling asleep at work. :-(Tina: I know what you mean. Tina: I keep on nodding off at my keyboard hoping that the boss doesn't notice.. Rita: The time just keeps on dragging on and on and on.... Rita: I keep on looking at the clock and there's still 4 hours of this drudgery to go. Tina: Times like these I really hate my work. Rita: I'm really not cut out for this level of boredom. Tina: Neither am I.	
Reference Summary (Ground Truth)	
Rita and Tina are bored at work and have still 4 hours left.	
Pre-trained Model	**Fine-tuned Model**
Tina and Rita have been talking about the boredom of working at the same job.	**Rita and Tina hate the boredom of their jobs.**

Table 7. Summaries generated for Test Example 2

6.6.3 Test Example 3

As for Dialogue 3 in Table 9, the fine-tuned summary showed improvement in both coherence and accuracy. While the pre-trained model introduced false information, incorrectly stating that the conversation is about "a friend" and the wedding happening "this weekend," the fine-tuned summary correctly identified that Eric was unsure about attending Ivan's brother's wedding due to obligations at home. The result even produced a coherent summary with transition word, ‘as’. At the same time, ROUGE scores also further proved the outstanding performance.

ROUGE 1	ROUGE 2	ROUGE L
53.06	38.30	48.98

Table 8. ROUGE performance of Test Example 3

Input Dialogue 3
Ivan: hey eric Eric: yeah man Ivan: so youre coming to the wedding Eric: your brother's Ivan: yea Eric: i dont know mannn Ivan: YOU DONT KNOW?? Eric: i just have a lot to do at home, plus i dont know if my parents would let me Ivan: ill take care of your parents Eric: youre telling me you have the guts to talk to them XD Ivan: thats my problem Eric: okay man, if you say so Ivan: yea just be there Eric: alright
Reference Summary (Ground Truth)
Eric doesn't know if his parents let him go to Ivan's brother's wedding. Ivan will talk to them.

Pre-trained Model	Fine-tuned Model
A friend of mine is getting married this weekend, and he has been talking to his brother about it.	**Eric is not sure whether to go to Ivan's brother's wedding as he has lots to do at home. Ivan will talk to Eric's parents.**

Table 9. Summaries generated for Test Example 3

6.7 Model Deployment

After finalising my model, I uploaded it to my Hugging Face repository to share my work with the broader data science community as an open-source resource. Once my model is published on Hugging Face, I can easily utilize it using the Hugging Face Transformers library's `pipeline` feature. This allows for integration into various applications. To use the model, I imported the `pipeline` function and specify the task, such as summarization (see attached supplementary code under "Testing.ipynb"). Users can generate summaries by passing input texts directly to the pipeline. This user-friendly interface enables quick experimentation with different texts and make the process highly accessible for both developers and researchers who are interested in exploring existing models.

To wrap up this project, I made my model accessible for others to use and build upon which hopefully could contribute to collaborative advancements in the field of NLP. This matches one of the objectives in this project: to provide a framework for summarizing conversational content from different domains in the future. To further improve, I could create a model card that details the model's architecture, dataset, and performance metrics to ensure transparency for users. This open-source initiative allows researchers and developers to explore and collaborate freely and fosters models' effectiveness in real-world applications.

7 Limitations and Future Work

In this project, several problems need to be addressed to improve the model's performance such as the limited variety of dataset, computational inefficiencies and linguistic bias in data. The BART model was only fine-tuned and tested on one set of dialogue dataset, SAMSum, where the domain and usage scenarios are limited to daily conversations. This may affect the model's generalization to a variety of real-life scenarios. Moreover, despite utilizing GPUs, the large model size required over an hour to merely complete six epochs. There is a need for more efficient optimization techniques or model compression strategies to reduce training time. Lastly, linguistic bias in the dataset poses a significant challenge. Gender bias, cultural bias, the use of different dialects etc. could lead to inaccurate text generation.

In the future development, I hope to expand the training and testing dataset such as using DialogSum for more real-life scenario dialogues (Chen et al., 2021), MediaSum for media interview dialogues (Zhu et al., 2021), and QMSum for query-based multi-domain meeting dialogues (Zhong et al.,2021). Besides, to address computational inefficiency issue, optimization techniques such as batch normalization and adjusting momentum can help models converge faster. Model compression like quantization, pruning and knowledge distillation can reduce the number of parameters and accelerate training (Cheng et al., 2017). As for bias elimination, it would be helpful to carefully select dataset with balanced representation of different user groups, expand to diverse language dataset and adopt bias mitigation techniques for a fairer and more representative outcome. I believe these could alleviate the shortcomings of this project.

Apart from mitigating the above limitations, I could further refine my current work by exploring more transformer models for comparison and analysis like Mistral AI, Meta Llama and more. Since I have uploaded my model to Hugging Face, it is my responsibility to creating model card that provides details on my model's architecture, dataset, training process and visualizations to ensure transparency for users in the next stage. This could facilitate better collaboration and continuous refinement of the model.

8 Conclusion

In this report, I have achieved my objectives of traiing a language model that generalized well on dialogue to provide coherent and concise summary as well as providing a framework for the advancement in dialogue summarization. In particular, I fine-tuned the state-of-the-art BART Transformer model as well as built a traditional RNN from scratch for dialogue summarization.

In comparing the two models, several architectural and technical differences were obvious. RNN relied on short-term memory processing for sequential processing, making it prone to losing long-range dependencies. In contrast, BART's self-attention mechanism process longer sequences by attending to all tokens simultaneously, which is crucial for handling complex dialogue structures and long sequences. The fine-tuned BART Transformer also benefits from its encoder-decoder architecture and pretraining to generalize across various types of texts, such as dialogues, let alone more structured forms like news articles and journals. These differences in attention mechanisms, pretraining, parallelism made BART significantly more effective than RNNs in summary generation.

As a result of Transformer model's architectural advantages, the final model produced accurate and coherent summaries with a high ROUGE score. It was trained and tested using a NVIDIA TESLA A100 GPU. After the training process, the training and validation loss reached a minimum of 1.26 and 1.46 respectively. In evaluation, 52.43% of ROUGE-1, 28.58% of ROUGE-2 and 44.22% of ROUGE-L were obtained. Human evaluation further supported the robust performance of the final model. Finally, I did not just work in a local development environment but actively took advantage of Hugging Face to share my work as an open-source project with the community.

As a whole, this project sets the groundwork for further development in dialogue summarization. Future work could include experimenting with more datasets, incorporating advanced optimization techniques, mitigating potential bias and exploring various transformer models. I value the process of exploring theories, mathematics, and the applications of deep learning in natural language processing. As I continue to explore the intersection of these two domains, this journey has enriched my understanding of how these theories and formula are translated into practical solutions for real-world problems. This year-long project ignited my passion for innovation in deep learning and NLP and motivated me to seek out new challenges in this domain.

Reference

Akter, S., Asa, A. S., Uddin, M. P., Hossain, M. D., Roy, S. K., & Afjal, M. I. (2017, February). An extractive text summarization technique for Bengali document (s) using K-means clustering algorithm. In *2017 ieee international conference on imaging, vision & pattern recognition (icivpr)* (pp. 1-6). IEEE.

Bahdanau, D., K. Cho, and Y. Bengio (2014). “Neural machine translation by jointly learning to align and translate”. In: arXiv preprint arXiv:1409.0473.

Chen, Y., Liu, Y., Chen, L., & Zhang, Y. (2021). DialogSum: A real-life scenario dialogue summarization dataset. *arXiv preprint arXiv:2105.06762*.

Cheng, Y., Wang, D., Zhou, P., & Zhang, T. (2017). A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*.

Cohan, A., Franck, D., Doo, S., Kim, T., Bui, S., Kim, W., Chang, N., Goharian, Research, A., & Jose, S. (2018). *A Discourse-Aware Attention Model for Abstractive Summarization of Long Documents*. https://arxiv.org/pdf/1804.05685.pdf

Devlin, J., Chang, M.-W., Lee, K., Google, K., & Language, A. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* (pp. 4171–4186). https://aclanthology.org/N19-1423.pdf

Ferreira, R., de Souza Cabral, L., Lins, R. D., e Silva, G. P., Freitas, F., Cavalcanti, G. D., ... & Favaro, L. (2013). Assessing sentence scoring techniques for extractive text summarization. *Expert systems with applications*, *40*(14), 5755-5764.

Floridi, L., & Chiriatti, M. (2020). GPT-3: Its nature, scope, limits, and consequences. *Minds and Machines*, *30*, 681-694.

Graves, A. (2012). Supervised sequence labelling. In Supervised sequence labelling with recurrent neural networks (pp. 5-13). Springer, Berlin, Heidelberg.

Harsoor, S. (2023). *Transformer model and variants of Transformer (chatgpt)*. Medium. https://pub.aimind.so/transformer-model-and-variants-of-transformer-chatgpt-3d423676e29c

Hernández-García, A., & König, P. (2018). Do deep nets really need weight decay and dropout?. *arXiv preprint arXiv:1802.07042*.

Islam, S., Elmekki, H., Elsebai, A., Bentahar, J., Drawel, N., Rjoub, G., & Pedrycz, W. (2023). *A COMPREHENSIVE SURVEY ON APPLICATIONS OF TRANSFORMERS FOR DEEP LEARNING TASKS*. https://arxiv.org/pdf/2306.07303.pdf

Jurafsk, D., & Martin, J. H. (2023). N-gram Language Models. https://web.stanford.edu/~jurafsky/slp3/3.pdf

Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., & Fidler, S. (2015). Skip-thought vectors. *Advances in neural information processing systems*, *28*.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., Zettlemoyer, L., & Ai, F. (2019). *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. https://arxiv.org/pdf/1910.13461v1.pdf

Lin, C.-Y. (2004). *ROUGE: A Package for Automatic Evaluation of Summaries*. https://aclanthology.org/W04-1013.pdf

Lipton, Z. C., Berkowitz, J., & Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*.

Liu, Y., & Lapata, M. (2019). Text summarization with pretrained encoders. *arXiv preprint arXiv:1908.08345*.

Luhn, H. P. (1958). The automatic creation of literature abstracts. *IBM Journal of research and development*, *2*(2), 159-165.

Maucher, J. (2022*). Sequence-To-Sequence, Attention, Transformer* – Machine Learning Lecture. Retrieved from https://hannibunny.github.io/mlbook/Transformer/attention.html

Piantadosi, S. T. (2014). Zipf's word frequency law in natural language: A critical review and future directions. *Psychonomic bulletin & review*, *21*, 1112-1130.

ROUGE - a Hugging Face Space by evaluate-metric. (n.d.). Huggingface.co. https://huggingface.co/spaces/evaluate-metric/rouge

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text Transformer. *Journal of machine learning research*, *21*(140), 1-67.

Saxena, P., & El-Haj, M. (2023). Exploring Abstractive Text Summarization for Podcasts: A Comparative Study of BART and T5 Models. *Proceedings of the Conference Recent Advances in Natural Language Processing - Large Language Models for Natural Language Processing*. https://doi.org/10.26615/978-954-452-092-2_110

Singh, F. (2023). *Fine-tuned BART for dialogue summary*. Kaggle. https://www.kaggle.com/code/farneetsingh24/fine-tuned-bart-for-dialogue-summary#Model-Evaluation-using-Rogue-Score

Smith, L. N. (2018). A disciplined approach to neural network hyper-parameters: Part 1--learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*.

Suleiman, D., & Awajan, A. (2020). Deep learning based abstractive text summarization: approaches, datasets, evaluation measures, and challenges. *Mathematical problems in engineering, 2020, 1-29. https://doi.org/10.1155/2020/9365340*

Sun, C., Qiu, X., Xu, Y., & Huang, X. (2020, February 5). How to Fine-tune Bert for long text classification? https://arxiv.org/pdf/1905.05583

Sutskever, I., Vinyals, O., V.Le, Q. (2014). *Sequence to Sequence Learning with Neural Networks*. https://papers.nips.cc/paper_files/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2- Paper.pdf

Torres, L. F. (2023). 📝 *text summarization with large language models*. Kaggle. https://www.kaggle.com/code/lusfernandotorres/text-summarization-with-large-language-models

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., ... & Scialom, T. (2023). Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Walker II, S. M. (n.d.). *What is the Rouge Score (recall-oriented understudy for GISTING evaluation)?*. Klu. https://klu.ai/glossary/rouge-score

Vaswani, A., Brain, G., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, Ł., & Polosukhin, I. (2017). Attention Is All You Need.
https://papers.nips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa- Paper.pdf

Yin, W., Kann, K., Yu, M., & Schütze, H. (2017). Comparative study of CNN and RNN for natural language processing. *arXiv preprint arXiv:1702.01923*.

Zargar, S. (2021). Introduction to sequence learning models: RNN, LSTM, GRU. *Department of Mechanical and Aerospace Engineering, North Carolina State University*.

Zhang, J., Zhao, Y., Saleh, M., & Liu, P. (2020, November). Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International conference on machine learning* (pp. 11328-11339). PMLR.

Zhong, M., Yin, D., Yu, T., Zaidi, A., Mutuma, M., Jha, R., ... & Radev, D. (2021). QMSum: A new benchmark for query-based multi-domain meeting summarization. *arXiv preprint arXiv:2104.05938*.

Zhu, C., Liu, Y., Mei, J., & Zeng, M. (2021). MediaSum: A large-scale media interview dataset for dialogue summarization. *arXiv preprint arXiv:2103.06410*.

Appendix

Model Architecture of BART:

```
BartForConditionalGeneration(
  (model): BartModel(
    (shared): Embedding(50264, 1024, padding_idx=1)
    (encoder): BartEncoder(
      (embed_tokens): Embedding(50264, 1024, padding_idx=1)
      (embed_positions): BartLearnedPositionalEmbedding(1026, 1024)
      (layers): ModuleList(
        (0-11): 12 x BartEncoderLayer(
          (self_attn): BartAttention(
            (k_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (v_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (q_proj): Linear(in_features=1024, out_features=1024, bias=True)
            (out_proj): Linear(in_features=1024, out_features=1024, bias=True)
          )
          (self_attn_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
          (activation_fn): GELUActivation()
          (fc1): Linear(in_features=1024, out_features=4096, bias=True)
          (fc2): Linear(in_features=4096, out_features=1024, bias=True)
          (final_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        )
    )
    (layernorm_embedding): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
   )
   (decoder): BartDecoder(
    (embed_tokens): Embedding(50264, 1024, padding_idx=1)
    (embed_positions): BartLearnedPositionalEmbedding(1026, 1024)
    (layers): ModuleList(
     (0-11): 12 x BartDecoderLayer(
      (self_attn): BartAttention(
       (k_proj): Linear(in_features=1024, out_features=1024, bias=True)
       (v_proj): Linear(in_features=1024, out_features=1024, bias=True)
       (q_proj): Linear(in_features=1024, out_features=1024, bias=True)
       (out_proj): Linear(in_features=1024, out_features=1024, bias=True)
      )
      (activation_fn): GELUActivation()
      (self_attn_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (encoder_attn): BartAttention(
       (k_proj): Linear(in_features=1024, out_features=1024, bias=True)
       (v_proj): Linear(in_features=1024, out_features=1024, bias=True)
       (q_proj): Linear(in_features=1024, out_features=1024, bias=True)
       (out_proj): Linear(in_features=1024, out_features=1024, bias=True)
      )
      (encoder_attn_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (fc1): Linear(in_features=1024, out_features=4096, bias=True)
      (fc2): Linear(in_features=4096, out_features=1024, bias=True)
      (final_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
     )
    )
    (layernorm_embedding): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
   )
  )
  (lm_head): Linear(in_features=1024, out_features=50264, bias=False)
)
```