A STUDY OF TEXT SUMMARIZATION WITH GRAPH ATTENTION NETWORKS

MOHAMMADREZA ARDESTANI
Bachelor of Science, University of Tehran Polytechnic, 2022

A thesis submitted
in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

A STUDY OF TEXT SUMMARIZATION WITH GRAPH ATTENTION NETWORKS

MOHAMMADREZA ARDESTANI

Date of Defence: August 2024

Dr. Yllias Chali Thesis Supervisor	Professor	Ph.D.
Dr. John Sheriff Thesis Examination Committee Member	Assistant Professor	Ph.D.
Dr. John Anvik Thesis Examination Committee Member	Associate Professor	Ph.D.
Dr. Wendy Osborn Chair, Thesis Examination Committee	Associate Professor	Ph.D.

Dedication

To my teachers, to whom I owe everything.

Abstract

This study aimed to leverage graph information, particularly Rhetorical Structure Theory (RST) and Co-reference (Coref) graphs, to enhance the performance of our baseline summarization models. Specifically, we experimented with a Graph Attention Network architecture to incorporate graph information. However, this architecture did not enhance the performance. Subsequently, we used a simple Multi-layer Perceptron architecture, which improved the results in our proposed model on our primary dataset, CNN/DM. Additionally, we annotated XSum dataset with RST graph information, establishing a benchmark for future graph-based summarizing models. This secondary dataset posed multiple challenges, revealing both the merits and limitations of our models.

Acknowledgments

I extend my deepest gratitude to Dr. Yllias Chali, whose continuous support and insightful guidance have been the cornerstone of my journey throughout this research. His expertise and encouragement have truly paved the road to my success.

I am immensely thankful to my friend and office mate, Wenzhao Zhu. The numerous, intellectually stimulating discussions we shared about the recent developments in the field of Natural Language Processing have been invaluable.

My sincere appreciation goes to the committee members, whose constructive feedback and fair evaluation significantly expedited my research progress. Their expertise and thoughtful critiques have been instrumental in refining my work.

I would also like to acknowledge the contribution of our summer internship students. Taha Abbass, who assisted in the segmentation of Elementary Discourse Units and the generation of Rhetorical Structure Theory graphs, has been a crucial part of our team. Additionally, Riya Saxena's involvement in graph preprocessing and the development of the Elementary Discourse Unit selection function has been vital to our project's success. Their dedication and hard work have left a lasting impact on this research.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Natural language processing (NLP) has various branches, namely Text-to-Text (T2T) Generation, Information Retrieval, and Machine Reasoning and Comprehension among others. One of the challenging tasks in T2T generation is Automatic Text Summarization. Text summarization has been studied for more than seven decades and has received a great deal of community attention over the last two decades (Rohil and Magotra, 2022), yet we are far behind the human performance in this task (El-Kassas et al., 2021).

Earlier developed systems were predominantly extractive (selecting part of documents as the final summary) while with the rise of Sequence-to-Sequence (seq2seq) models they leaned towards being abstractive (Hou et al., 2018), capable of paraphrasing for the final summary. One of the successful variations of seq2seq approaches is Transformer (Vaswani et al., 2017), initially designed for Machine Translation, which has contributed to higher performance in various language generation tasks. With the notion of key, query, and value, it generates a relatively rich intermediate representation. Although token[1] level and shallow, this representation is good enough, based on the experimental results, for machine translation and performs near human level and even outperforms humans in some specific circumstances (Popel et al., 2020).

Transformers and their variation, however, fall short in more cognitively complicated language generation tasks, namely Dialogue Management or Text Summarization in which

[1] In NLP, a token is a basic unit of text, such as a word, subword, or character, used in processing. Tokenization breaks down text into these units for analysis by models.

we have to manage long dependencies, reason over the importance of sentences, and how tailor important parts coherently and cohesively. Zhu et al. (2021) showed that enriching previous summarization models with graph data structures, particularly entity-relation information graphs, can increase their factual consistency. Moreover, Xu et al. (2020) showed that using a Rhetorical Structure Theory (RST) graph and Co-reference (Coref) graph can improve the performance of summarization systems. Following the later study, we also incorporate RST and Coref graphs in the encoder part of our model and leverage, in part, Graph Neural Networks (GNNs) to incorporate the information of graphs.

A recent study (Liu and Wu, 2022) shows five versatile graphs (text graphs, syntactic graphs, semantic graphs, knowledge graphs, and hybrid graphs) can significantly improve the performance of models in T2T generation tasks. Despite this finding, there are very few graph datasets for Text Summarization. We provide the community with one reprocessed graph dataset to bridge this gap.

1.2 Contribution

The key contributions of our research are outlined as follows:

- Provided a graph-annotated version of XSum (Narayan et al., 2018), with clear steps for annotating other datasets as well.
- Developed a stage-wise summarization model that is capable of processing lengthy documents and utilizes various optimization techniques in its fine-tuning phase.
- Incorporated graph structures to improve the performance of our baseline models using Multi-layer Perceptron architecture.

1.3 Thesis Organization

The subsequent chapters are organized as follows. In Chapter 2, we review the literature, mostly employing a top-down approach of describing the concepts. Next, we discuss related

works in Chapter 3, followed by Chapter 4 which introduces an auxiliary dataset that serves as a secondary benchmark to test the robustness of our approach. This chapter details the preparation of the dataset, including all related tools and terminologies.

Chapter 5 delves into our proposed approach, outlining its parameters and hyperparameters which are used in training, fine-tuning, and inference time. This chapter also provides justifications for our design decisions and details the experimental setups for replication purposes. Chapter 6 presents our results and discusses them from various perspectives. Finally, Chapter 7 concludes the thesis, offering a summary and suggesting potential areas for future research.

Additional information that aids in clarification, but which could divert the reader if included in the main chapters, is added to the Appendix.

Chapter 2

Background

The field of computer science encompasses numerous sub-branches, each of which has received varying levels of contributions over time. Artificial Intelligence (AI) is one notable branch that has been at the center of attention for recent years. What we call AI today, is technically considered Artificial Narrow Intelligence (ANI) (Kalota, 2024). ANI involves combining mathematical and statistical methods with programming techniques to create systems specialized in specific domains, aiming to perform at or near human levels but they may also outperform humans in a specific task. Moreover, there are two notable extensions of AI that differ in their domain and level. Artificial General Intelligence (AGI) seeks to match human performance across all domains, while Artificial Super Intelligence (ASI) aims to develop systems that surpass human capabilities in all domains (Kalota, 2024).

Among the diverse areas of AI, Natural Language Processing (NLP) stands out as a critical field. NLP delves into the intricate relationship between computer systems and human language, emphasizing the development of algorithms that enable computers to process, interpret, and analyze substantial volumes of natural language data. A noteworthy application within NLP is text summarization, which aims to distill the essential information from a larger body of text into a concise and informative summary, maintaining the original text's context and meaning (El-Kassas et al., 2021).

Our effort is concentrated on improving current text summarization systems, specifically by using Large Language Models (LLMs) and Graph Neural Networks (GNNs). There are other subtleties to these main topics that will be discussed throughout this chapter.

Fig 2.1 demonstrates the connection between Computer Science and our topic of study.

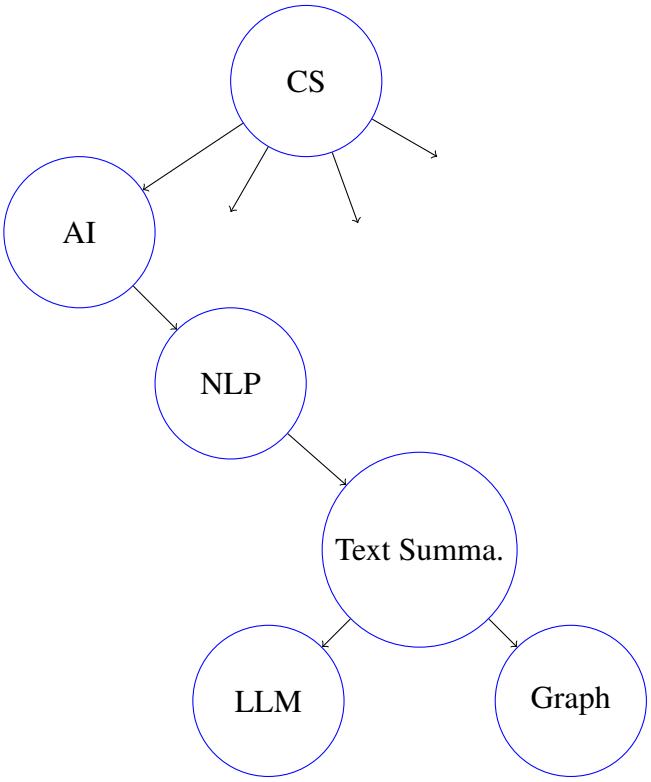


Figure 2.1: Rough taxonomy chart of our work.

2.1 Neural Networks

Artificial Neural Networks, often called Neural Networks (NNs), are a powerful approach that serves as a building block for many AI-related applications and academic research, including ours. In the following subsections, we discuss important constituent elements of this approach.

2.1.1 Single Perceptron

The idea of neural networks has been in existence for many decades, taking its roots from neuroscience in biology. Anderson and McNeill (1992) made a comparison between biological neurons and those in computer science. This comparison is depicted in Figure 2.2, which shows the Artificial Processing Elements, often simply referred to as neurons or

perceptrons. These elements take some inputs to perform a process, resulting in an output. There is also an arrow that loops back to the perceptron, signifying feedback information. This is similar to a specific neural network architecture used today, the Recurrent Neural Networks (RNN). However, this feedback arrow is not considered an essential part of the simple perceptron model. Instead, we add one bias to the summation. A single processing unit, or perceptron, is often adequate for simple tasks, but for more complex challenges, we need to stack multiple perceptrons, called Multi-Layer Perceptron (MLP).

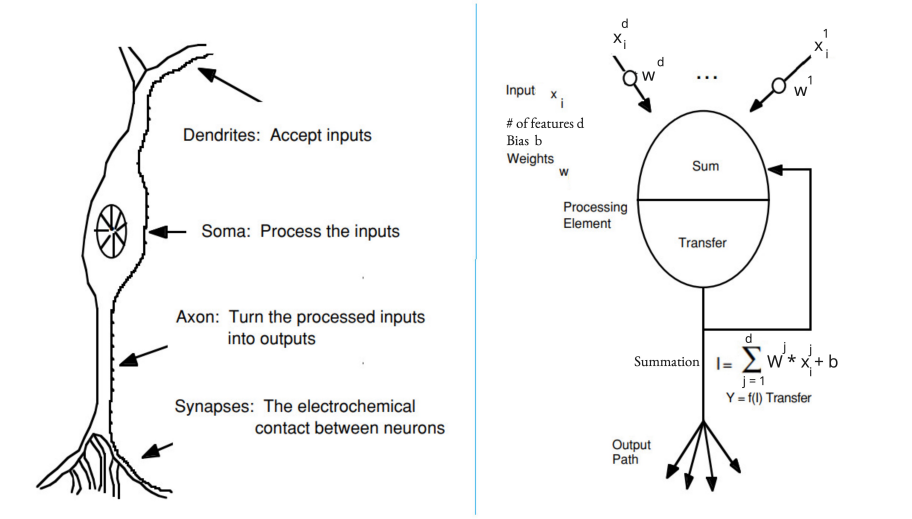


Figure 2.2: A simple biological neuron and artificial neuron (perceptron), adapted from (Anderson and McNeill, 1992).

We will elaborate on the notations of a single perceptron unit, which will be used in the following sections. Each unit is capable of receiving N inputs each having d features. Then, for each input, the perceptron unit multiple each feature of each input data and possibly add one bias term in the summation. Finally, the summation will be passed to a transformation function, also refereed to as activation function, to generate the final output for each data point. More formally, a dataset is written as a set of example-label pairs $\{(x_1, y_1), \ldots, (x_n, y_n), \ldots, (x_N, y_N)\}$. The table of examples $\{x_1, \ldots, x_N\}$ is often concatenated, and written as $X \in \mathbb{R}^{N \times D}$, similarly $Y \in \mathbb{R}^N$, and the i^{th} input data with d features is

a column vector as $x_i = (x_i^1, x_i^2, \ldots, x_i^d)^\top$. In order to calculate y_i from the input we have:

$$
\begin{aligned}
y_i &= Activation \left(\begin{pmatrix} & & \\ \left(w^1, w^2, \ldots, w^d\right) & & \\ & & \end{pmatrix} \begin{pmatrix} x_i^1 \\ x_i^2 \\ \vdots \\ x_i^d \end{pmatrix} + b \right) \\
&= Activation \left((w^1, w^2, \ldots, w^d) \cdot (x_i^1, x_i^2, \ldots, x_i^d)^\top + b \right) \\
&= Activation \left(W \cdot x_i + b \right)
\end{aligned}
$$

And for Y, output of all N input data points, we have:

$$Y = Activation(\ W \cdot X^\top + b\ )$$

Where $X \in \mathbb{R}^{N \times D}$ is the matrix of features for all N examples, $W \in \mathbb{R}^D$ is the weight vector, $b \in \mathbb{R}$ is the bias term (broadcasted across all N examples). Now we are able to use this notation for explaining the concept behind a perceptron unit.

To clearly explain this section's concept, we chose a concrete example that can be modeled with a single perceptron. Let's say we want to model the relation between the temperature of a certain metal and its length and we have this meta-knowledge that the polynomial that represents this relation in the ideal situation is linear; hence, we can choose a single perceptron with a linear transformation function (like identity function) to model this problem, learn the relation, or in other words, estimate the parameters. First, we perform some experiments and calculate the length under different temperatures. These data points will have some noise due to the nature of the experimental work. Figure 2.3, drawn by matplotlib library[1], illustrates 5 simulated data points from the ideal line by adding noise from the normal distribution $N(\mu, \sigma^2) = (0, 1)$.

[1] https://matplotlib.org/

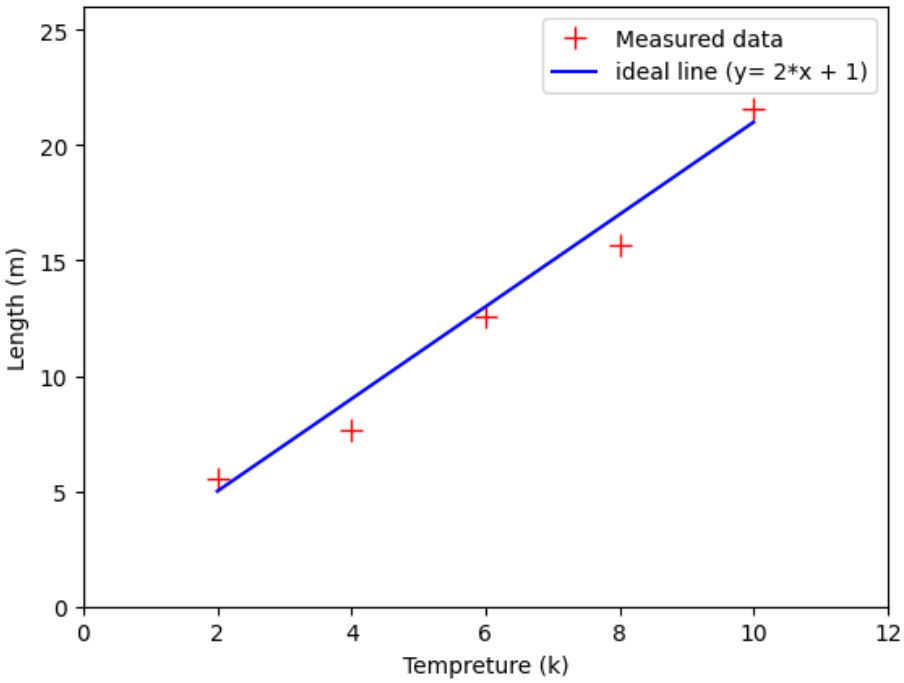


Figure 2.3: Points are: $\{(2, 5.51), (4, 7.63), (6, 12.55), (8, 15.63), (10, 21.55)\}$

This example provides a deep analogy to our text summarization task as well. Instead of the linear function, we have a complex function that represents the mapping between input documents and their gist. Ideally, this complex function is the best function that considers all the aspects of natural language, like length constraints, style, grammar, etc. However, we do not have all the meta-knowledge to know the format of this function and the number of parameters it has, unlike the simple length-temperature problem. As a result, researchers usually pick a large neural network that can be at least as large as the ideal function, although we might have a higher cost in training that network. Similarly, we first collect sample data of documents and summaries. These data points, for sure, have their own noise but we hope that we can have a good enough estimate from the ideal function based on them.

We can find the global optimal solution of our simple example in an analytical way. However, analytical approaches fall short as the problem scales. Therefore, numerical approaches are preferred and neural networks are one of them. In the following subsections,

we will discuss the steps to estimate the parameters (w and b) using a neural network mind-set.

2.1.2 Activation Function

As mentioned earlier, we can choose an identity function as the Transfer Function, also known as the Activation Function, for our task of modeling a line. For any non-linear function, however, we cannot estimate the model without any non-linear transformation. Therefore, many non-linear activation functions are proposed such as sigmoid function (Narayan, 1997), ELU (Clevert et al., 2016), or Gelu (Hendrycks and Gimpel, 2023).

2.1.3 Forward Propagation

Now that we have the architecture of our neural network, we can pass the X values to the network to calculate the predictions (noted as $\hat{y}$). This process is called forward propagation. We need to know that at the beginning of the estimation of the best values of the parameters, we do not have any initial value for these parameters. Thus, we need to first initialize them, possibly randomly. There are better methods, however, to initialize weights. By initializing $w = 1$, $b = 1$ and using Identity activation (shortened as ID) we will have:

$\hat{y} = ID(W * X^\top + b) = ID([1] * [2,4,6,8,10] + 1) = [2,4,6,8,10] + 1 = [3,5,7,9,11]$

in which the bias term is broadcasted.[1]

2.1.4 Loss and Optimizers

Clearly, the predictions ($\hat{y}$) that we made in the previous subsection, are far from the actual values (y). We need to define this error numerically so that we can measure and minimize it later. The loss function, also known as the Criterion, indicates how much error we have in our predictions. Mean Square Error (MSE) is among the many task-specific loss

[1] Broadcasting in matrix operations allows for efficient computation on matrices of different shapes without requiring explicit reshaping. For example, if we add the scalar 1 to the vector $[2,4,6,8,10]$, matrix multiplication libraries will internally expand this scalar to the vector $[1,1,1,1,1]$ and perform the addition. This process, known as broadcasting, simplifies operations and enhances memory efficiency.

functions that exist by which we try to minimize the error and, consequently, find the best parameters.

MSE Loss (Y, X, W, b) $= \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(Y_i - (W * X^\top + b))^2$

By using the values of y and $\hat{y}$ we can calculate the value of the loss function to ensure that the model is minimizing it but we do not need this value for the optimization process per se.

Now, that we have the loss function and initial values for parameters, we can update the weights using Gradient Decent Algorithm (Ruder, 2017). All the optimization algorithms for neural networks are based on this approach but they might vary in the details as to how they deal with local minimum issues.

2.1.5 Backward Propagation

Backpropagation, also known as backward propagation, is a vital mechanism for training neural networks (Deisenroth et al., 2020). It is particularly important for optimizing the network's weights during training. After the input data goes through the network layers to produce an output in the forward pass, the backward propagation process begins. This process involves calculating the gradient of our loss function with respect to the neural network's parameters (weights) using the chain rule of calculus. These gradients indicate how much and in which direction the weights should be adjusted to minimize the loss. This is critical for optimization because it provides a systematic way for the learning algorithm to learn from the network's errors. Without backpropagation, the network would not be able to update its weights in a way that minimizes the loss, leading to an inability to improve performance or learn essential patterns in the training data. Essentially, backpropagation simplifies the complex task of optimizing thousands or even millions of weights in a deep neural network, making effective training of deep networks possible.

2.1.6 Regularization

As we alluded to earlier, for real-world problems, we do not know how many parameters are required to model the task. So, if we acquire a complex function we might just memorize the train data set without learning the general structure of the data. This issue is called overfitting in Machine Learning. Using our simple example, if we choose a polynomial of degree 4 and then optimize the coefficients, then we will most likely just memorize the data because we can always fit a polynomial of degree n to n + 1 numbers. This is a classic mathematical problem in curve fitting, which is often solved using polynomial regression (Pourkarimi et al., 2011). Figure 2.4 shows a polynomial of degree 4 that fits our data. This estimation is done by using NumPy library[1].

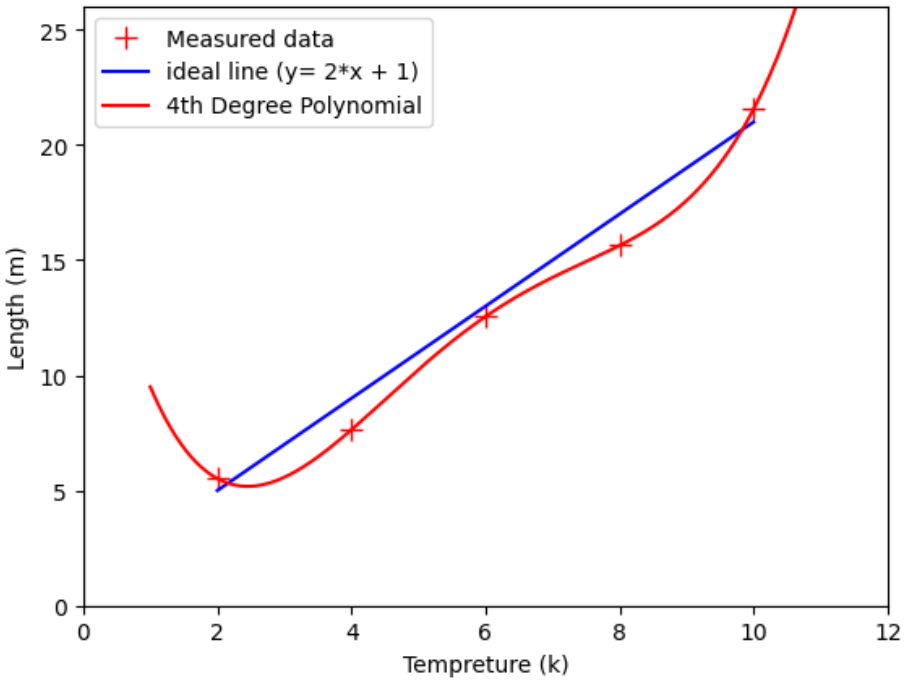


Figure 2.4: Overfitting issue.

To tackle this concern, regularization methods are introduced. They generally try to penalize the use of more parameters or learning the noise of data. Amongst many regularization techniques, we can refer to L2 regularization (van Laarhoven, 2017) and Dropout

[1] https://numpy.org/

(Srivastava et al., 2014). A dropout layer randomly "drops out" or deactivates a fraction of neurons in the layer during training, forcing the network to learn more robust features by not relying on any single neuron. This helps improve the model's generalization to new data.

2.2 Language Models

NLP encompasses many tasks, including word representation, relation representation, and named entity recognition. One essential task within NLP is language modeling, which plays a crucial role in several applications like speech recognition, machine translation, and text summarization. In the following subsection (2.2.1), which is based on the Speech and Language Processing book (Jurafsky and Martin, 2000), we define the language modeling tasks. Afterwards, we introduce one specific type of it. Then, we complete this section with elaborating on the recent architectures for this task.

2.2.1 Language Modeling Tasks

Language modeling is the task of predicting the probability distribution of a sequence of words in a language. It involves estimating the likelihood of various word sequences, typically to predict the next word in a sentence based on the previous words. This task is foundational in NLP and underpins many applications, as mentioned earlier. Mathematically, a language model is an approach that estimates the probability of a sequence based on its words such that: $P(S) = P(w_1, w_2, ..., w_i, ..., w_n)$. For example, if we hear a talk and we are not sure whether the person said "Some dolphins can recognize speech" or "Some dolphins can wreck a nice peach" our mind tends to opt for the former by attributing more probability to it, on account of being naturally trained on a variety of English content. We can also program computers to find the best probability function that resonates with human judgment on this task.

N-grams are a fundamental concept in natural language processing, acting as simple yet

effective language models. An n-gram is a sequence of 'n' items (words, letters, syllables, etc.) from a given text. In the context of language models, these items are typically words. The basic idea behind using n-grams as language models is to predict the probability of a word given the preceding sequence of words.

The conditional probability of an n-gram can be formulated as the probability of a word occurring, given the preceding sequence of words. For a bigram (2-gram) model, which considers pairs of words, the formula is:

$P(w_n|w_{n-1}) = \frac{P(w_{n-1},w_n)}{P(w_{n-1})}$

Here, $P(w_n|w_{n-1})$ is the probability of the word w_n occurring, given the previous word w_{n-1}. This is calculated by dividing the joint probability of the sequence (w_{n-1}, w_n) by the probability of the preceding word w_{n-1}.

For a trigram (3-gram) model, the formula extends to:

$P(w_n|w_{n-2}, w_{n-1}) = \frac{P(w_{n-2},w_{n-1},w_n)}{P(w_{n-2},w_{n-1})}$

This represents the probability of w_n given the two preceding words (w_{n-2}, w_{n-1}). In practice, these probabilities are usually estimated from a corpus of text by counting occurrences. However, n-gram models have limitations, such as handling unseen word sequences (zero probability issue) and growing computational complexity as 'n' increases. techniques like smoothing and back-off methods are often used to address these challenges.

2.2.2 Neural Language Models

Having defined the concept of Language Modeling and introduced a simple approach through n-grams, we now transition to the core concept behind the prevailing methodologies in modern computational linguistics: neural-based modeling. This shift marks a significant advancement from traditional statistical models to more sophisticated architectures capable of capturing complex language patterns. By Figure 2.5, we bring an illustration of a simple Neural Network (NN) model that predicts the next word in a sequence given its previous words.

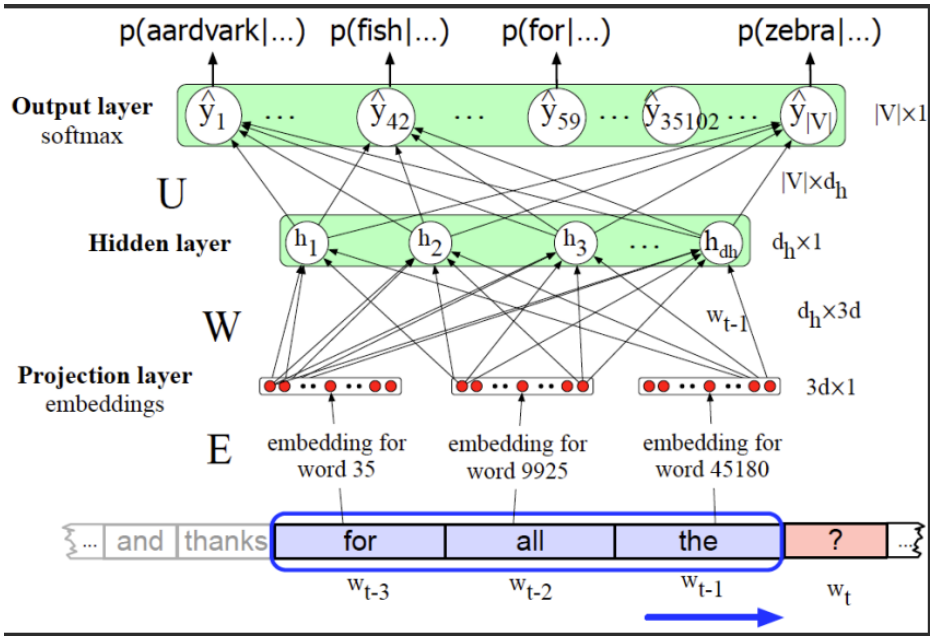


Figure 2.5: MLP designed for language modeling, from (Jurafsky and Martin, 2000). V is the vocabulary size and embeddings are one-hot vectors for words.

2.2.3 Large Language Models

Recent advancements in algorithms and the availability of high-performance computation power have allowed researchers to build upon the simple neural networks, presented in the previous subsection (2.2.2), capable of capturing complex language patterns. These Large Language Models (LLMs) have revolutionized researchers ability to model and understand natural language by leveraging large datasets and intricate network structures.

BERT (Devlin et al., 2019) stands for Bidirectional Encoder Representations from Transformers, which is one of these revolutionized LLMs. It is called bidirectional because it processes text in both directions simultaneously by using the attention mechanism (Vaswani et al., 2017). BERT utilizes only the encoder part of the Transformer architecture (Vaswani et al., 2017), focusing on encoding text information rather than decoding or generating text. The model is pre-trained, a practice that involves using a large amount of data to make the model an expert in a domain, in our case natural language (NL), allowing it to be used with minimal modification for various sub-domain tasks within NL. Given the complexity of NL, BERT required a substantial model size; thus, the BERT-base model features 12 lay-

ers. For pre-training, it used an enormous corpus of unlabeled text, specifically the English Wikipedia and the BooksCorpus (Zhu et al., 2015). The pre-training tasks employed were Next Sentence Prediction (NSP) and Masked Language Modeling (MLM), which trained the model to understand language context and structure deeply. All the pre-trained LMs are common in these three mentioned components; (1) large architecture, (2) massive training data, and (3) multiple domain-related training tasks. After applying these steps and training the model, we can use it for other sub-domain tasks (also known as downstream tasks) with minimal training (called fine-tuning). Figure 2.6 illustrates how Devlin et al. (2019) tested the model by fine-tuning it on eight downstream tasks and all of them outperformed the previous SOTA by a large margin.

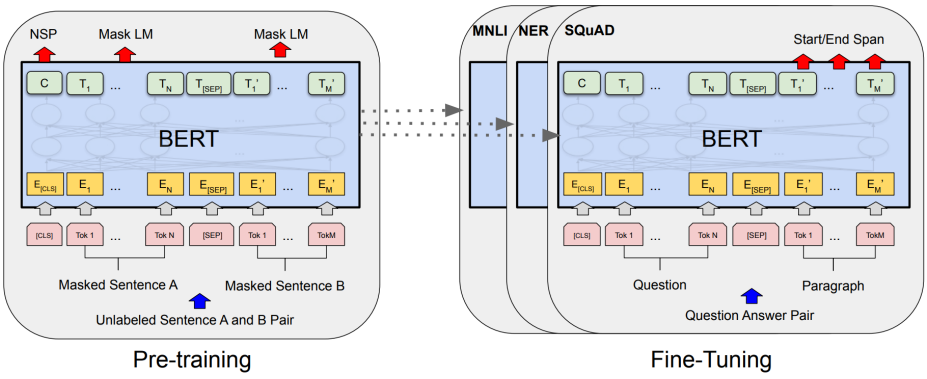


Figure 2.6: BERT architecture, from (Devlin et al., 2019).

The Longformer (Beltagy et al., 2020), standing for long Transformer, introduces a pivotal solution to the challenge of processing extensive textual data. Conventional Transformer-based architectures calculate the self-attention[1] vector for each token based on all other tokens in the input sequence. Consequently, if the input sequence contains n tokens, the self-attention operation has a time complexity of $O(n^2)$, making it computationally demanding. This quadratic scaling of memory and computational demands limits the ability of these architectures to handle long sequences effectively (Beltagy et al., 2020). The innovative architecture of the Longformer is characterized by its attention mechanism that increases

[1]Attention allows a model to focus on different parts of the input when making predictions, assigning different weights to different elements based on their relevance. Self-Attention is a specific type of attention mechanism where the model applies attention to different positions within the same sequence.

linearly with the length of the sequence, thereby enabling the analysis of substantially larger documents than previously possible. This model achieves efficiency through a strategic blend of localized attention within small windows and global attention for selected tokens across the entire text. Figure 2.7 demonstrates three new attention techniques employed by Longformer. Such an approach ensures not only the efficient handling of large-scale documents but also the preservation of the model's ability to discern and interpret critical long-range dependencies within the text. This methodology significantly enhances the model's utility across various NLP applications requiring comprehensive contextual understanding over extensive textual segments, including, but not limited to, document summarization, detailed question answering, and sophisticated document classification tasks.

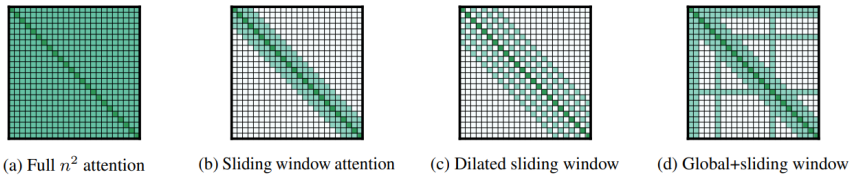


Figure 2.7: Novel attention mechanism introduced by Longformer (Beltagy et al., 2020). Green color indicate whether a token is attending to other tokens in the input sequence or not.

2.3 Graph Neural Networks

2.3.1 Necessity

Current neural network models are specialized in working with simple sequences or grids (Hamilton, 2020). Figure 2.8 compares the structure of representation of image, audio and text into simple graphs as well as the representation of other data formats, such as molecules, in a complex graph.[1] Most of the data that were fed into NNs to learn were text, image, or audio based. This does not entail that most of the real-world data are or can be represented in these three formats, however (Wu et al., 2022). In fact, there are

[1] In Fig 2.8, the image of graph is licensed under Creative Commons (https://en.wikipedia.org/wiki/File:Atisane3.png.) and the image of 3 is under Creative Commons (https://www.tensorflow.org/datasets/catalog/mnist.). The other images in this collage are made by us.

numerous applications that are naturally or best represented in a graph format, including, molecular anatomy, information, knowledge, brain/neurons, genes, social media, and many more. Although in our research we are working with text that can be represented in a simple sequential format, we can have a graph-based representation of it as well. We will elaborate on this in Chapter 4.

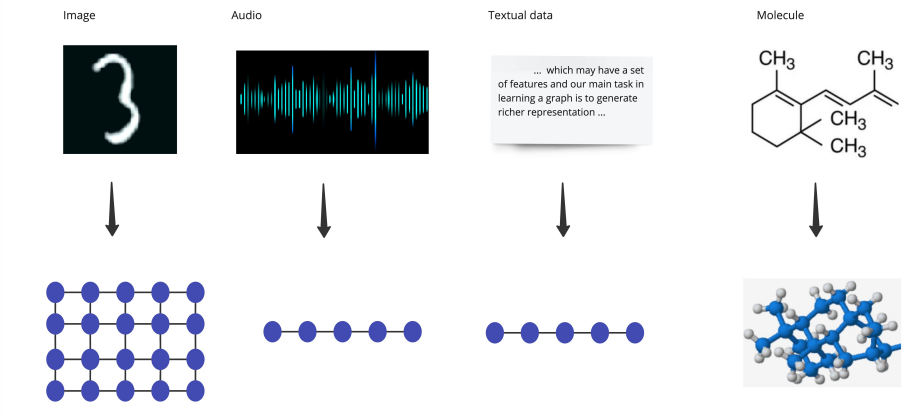


Figure 2.8: Simple grid/sequence representation versus complex graph representation as the data form. The illustration is by this author, and the idea is inspired by (Hamilton, 2020).

GNNs enable predictions at three distinct levels: node, edge, and graph (Hamilton, 2020). At the node level, GNNs can classify individual nodes, such as identifying fraudulent transactions in a financial network. For edge prediction, they assess the likelihood of a relationship between two nodes, exemplified by predicting potential collaborations between researchers within an academic network. At the graph level, GNNs evaluate entire structures, such as determining the pharmacological activity of chemical compounds. Our focus primarily lies on node-level applications, harnessing GNNs to classify nodes of a graph that represent a document.

2.3.2 Techniques

Graphs consist of two main components that are nodes and edges each of which may have a set of features and our main task in learning a graph is to generate richer representation (embedding) for each node so that we can later use them for the aforementioned

three-level predictions. Current GNN architectures meet this objective by two operations; Message aggregation and Update. Figure 2.9, illustrates these two operations.

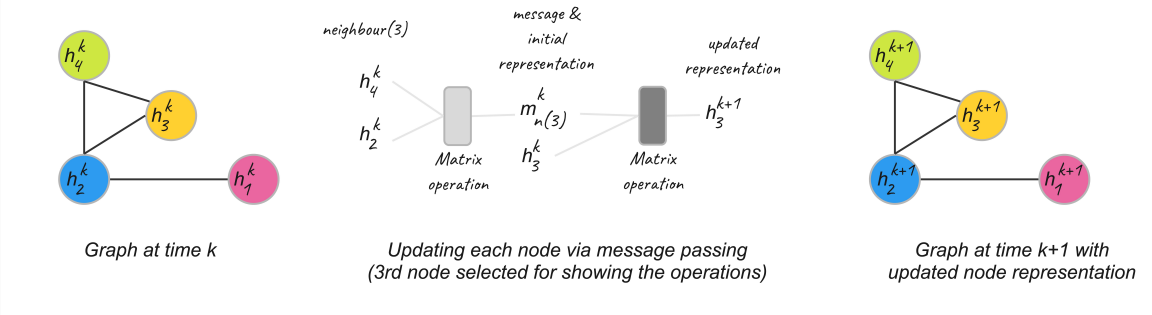


Figure 2.9: Message aggregation and update in GNNs for a single node from its local neighborhood

During each iteration of the message-passing phase in a GNN, the hidden embedding $h_u^{(k)}$ for each node $u \in V$ is updated based on the information aggregated from u's neighborhood, denoted as $N(u)$. This update process during message passing can be expressed as follows:

In every round of the message-passing stage within a GNN, the hidden representation $h_u^{(k)}$ associated with every node u that belongs to the vertex set V undergoes modification. This modification leverages the collective information sourced from the neighborhood of u, symbolized as $N(u)$. The method for updating the node's embedding during this message-passing sequence is formulated as Equations 2.1 and 2.2.

$$h_u^{(k+1)} = \text{UPDATE}^{(k)}\left(h_u^{(k)}, \text{AGGREGATE}^{(k)}\left(\{h_v^{(k)}, \forall v \in N(u)\}\right)\right) \tag{2.1}$$

$$= \text{UPDATE}^{(k)}\left(h_u^{(k)}, m_{N(u)}^{(k)}\right), \tag{2.2}$$

UPDATE and AGGREGATE represent distinct differentiable functions, such as neural networks, and $m_{N(u)}$ denotes the aggregated "message" from the neighborhood $N(u)$ of node u. Superscripts are employed to differentiate between the embeddings and functions across various message-passing iterations. Furthermore, these iterations of message passing are often referred to as the different "layers" of the GNN, which means they do not

necessarily have the same weights.

Different GNN techniques differ in the way that they define these two fundamental functions. Two widely used architectures are Graph Attention Networks (GATs) (Veličković et al., 2018) and Graph Convolutional Networks (CGNs) (Kipf and Welling, 2017).

2.4 Related Concepts to Dataset Preparation

In the following sections, we will delve into various concepts and tools integral to the dataset processing pipeline, addressing the "what", "why", and "how" aspects for each. "What" entails defining and introducing each concept briefly. "Why" delves into the rationale behind their incorporation in our workflow, while "how" elucidates the approach to implement or use these concepts and tools.

2.4.1 Elementary Discourse Unit

What and Why

Natural text is typically a collection of sentences, but this is not always the case. In many extractive approaches, such as Ruan et al. (2022), sentences are treated as the fundamental unit in the selection process. However, not every part of a sentence provides important information for summarization. For instance, in the sentence "The meeting was rescheduled, which was not surprising to anyone, and it will now be held tomorrow." we identify three parts, each separated by commas. Here, the second part is peripheral. To avoid selecting the whole sentence, which might have redundant information, we use EDUs. An EDU is a segment of text that conveys a single, coherent piece of information within a larger discourse that is an alternative unit of selection.

A useful guideline is that if an EDU provides only supplementary information, it can be omitted without losing key content. Table 2.1 , using document id '1' from our processed XSum dataset, illustrates how decomposing a sentence into its EDUs can help avoid selecting redundant information. For example, in a sentence decomposed into three EDUs, we

might only select the first and last EDUs, omitting the middle one if it offers extra information that is not as crucial as the others. The details of this selection process will be further discussed later in this chapter.

Table 2.1: Important EDU Selection Sample.

Original Text	Officers searched properties in the waterfront park and colonsay view areas of the city on Wednesday. Detectives said three firearms, ammunition, and a five-figure sum of money were recovered. A 26-year-old man who was arrested and charged appeared at Edinburgh Sheriff Court on Thursday.
EDU Segmentation	"officers searched properties in the waterfront park and colonsay view areas of the city on Wednesday." "detectives said" "three firearms, ammunition and a five-figure sum of money were recovered." "a 26-year-old man" "who was arrested and charged" "appeared at Edinburgh Sheriff Court on Thursday."
Ground Truth Summary	a man has appeared in court after firearms, ammunition and cash were seized by police in Edinburgh.

Highlighted EDUs are labeled as important.

The authors of the recent paper, (Xu et al., 2020), conducted a precise numerical analysis to determine if EDUs are a better unit of selection than sentences. Comparing the ROUGE scores of these two units using the same selection approach, their analysis spanned the entire CNN/DM dataset. The results of this comparison are presented in Table 2.2.

Table 2.2: Numerical Comparison of ROUGE Scores for EDUs and Sentences as the Selection Unit in CNN/DM Dataset.

Type	**R-1**	**R-2**	**R-L**
Sentence	55.61	32.84	51.88
EDU	**61.61**	**37.82**	**59.27**

The notion of Elementary Discourse Units (EDUs) was first introduced in Rhetorical Structure Theory (Mann and Thompson, 1988) as the unit of analysis. However, its application has extended beyond this original domain. Now, EDUs are used as the unit of extrac-

tive summarization, as well as in different types of trees or graphs, such as Co-reference graphs. The underlying idea is to chunk a sentence down into discourse units that each convey a complete meaning. We elaborate on this process in the following subsection.

How

We review NeuralEDUseg (Wang et al., 2018) a recent paper that can perform EDU segmentation. NeuralEDUseg proposed model reached 94.3 F1 score in segmentation, which is a high score for this task.

We illustrate the NeuralEDUseg model in Figure 2.10. Broadly, the model employs a Bidirectional Long Short-Term Memory (BiLSTM) (Schuster and Paliwal, 1997) combined with a Conditional Random Field (CRF) framework (Lafferty et al., 2001). This approach sets up discourse segmentation as a sequence labeling task, wherein each word within a sentence is labeled as either the start of an EDU or not.

The model employs word embeddings, specifically GloVe embeddings (Pennington et al., 2014), as part of its input features, indicated by e_i in the figure. To address the issue of data insufficiency and enhance the model's ability to capture complex features, the model incorporates ELMo (Embeddings from Language Models) (Peters et al., 2018) representations, which is marked by r_i in the figure. These representations are pre-trained on a large corpus and are known for capturing both semantic and syntactic nuances of words.

The article uses a limited self-attention mechanism designed to effectively model critical long-range dependencies essential for detecting EDU boundaries while also diminishing the influence of irrelevant information, enhancing the model's proficiency in recognizing pertinent context signals vital for EDU segmentation. This approach concentrates on a specific window surrounding each word instead of the full text. Based on their result, the best value for this hyperparameter, window size, is 5, which results in 94.3 F1 score. Another observation is that if we consider a very large window size, the F1-score decreases to 93.8 value.

The model is trained and tested on the RST-DT (Rhetorical Structure Theory Discourse Treebank) corpus (Carlson et al., 2001). Various hyperparameters, including the window size for the restricted attention mechanism and the dimensions of the embeddings, are tuned for optimal performance.

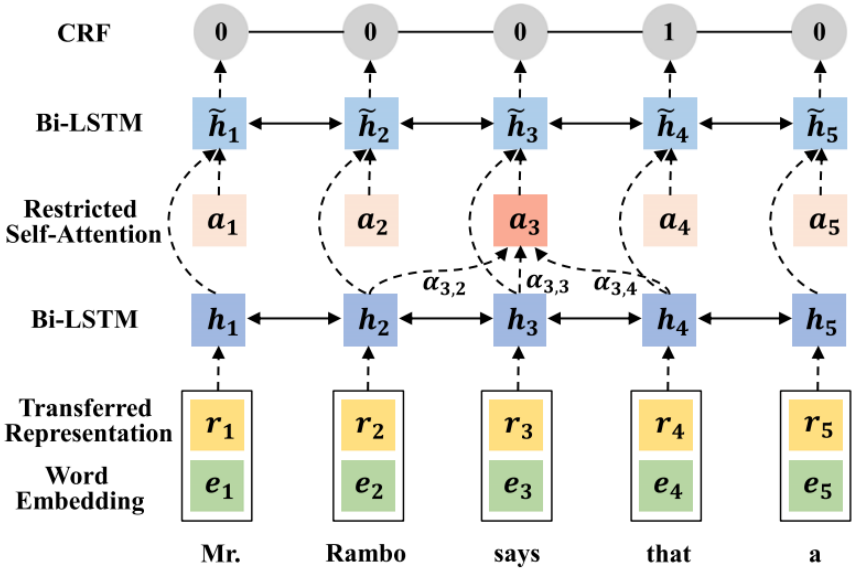


Figure 2.10: NeuralEDUseg Model illustration, from (Wang et al., 2018).

2.4.2 Rhetorical Structure Theory

What and Why

As we discussed previously (Section 2.3), graph structures are widely used across various domains for representing data, and NLP is no exception. Rhetorical Structure Theory, which was proposed in the 1980s by Mann and Thompson (1988), is a method for examining the structures of text. It focuses on how different sections of a text are interconnected, resulting in a tree representation of the text. Grounded in discourse analysis, this theory aims at deciphering the cohesive links between text segments, thereby forming an organized entity. It suggests a hierarchical arrangement of text elements, termed "spans", each with a distinct role relative to others and linked by rhetorical connections such as contrast, cause-effect, elaboration, or justification. RST is invaluable for dissecting complex text

structures and is utilized in linguistics, natural language processing in computer science, and the development of automated text summarization tools. By scrutinizing the rhetorical relationships within a text, RST seeks to uncover the author's purpose and their strategies for persuading or informing the reader.

The terminology and methodology of the proposed RST have been modified and expanded over the last two decades since it was first introduced. For instance, Xu et al. (2020) introduced an extension to convert the RST tree into an RST graph. This approach linearizes the tree, making it easier for neural networks to be incorporated, while still preserving its long-range document-level connections between EDUs. We also utilize these RST graphs in our approach. The Co-reference graph offers an alternative to RST, providing a graph representation of the original text, a topic we elaborate on in Section 2.4.3. Other alternatives include, but are not limited to, Graph bank (Wolf and Gibson, 2005) or PDTB (Miltsakaki et al., 2004). We naturally read and comprehend documents in a graph structure. For example, we organized this document into chapters, sections, and subsections so that readers can more easily comprehend or summarize the content for themselves.

Fig 2.11, is a modified example from (Mann and Thompson, 1988) to illustrate the RST tree and graph. In this diagram, in which the second EDU is supposed as a satellite node and the others as nucleus nodes, we have the following text:

1. The next music day is scheduled for July 21 (Sunday), noon-midnight.
2. I'll post more details later,
3. but this is a good time to reserve the place on your calendar.

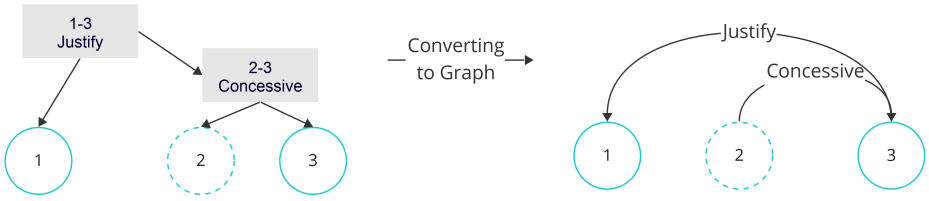


Figure 2.11: RST Tree and Graph Illustration with Edge Types.

Concepts in Rhetorical Structure Theory include:

1. **Textual Segments**: These are either a span or an EDU. A span is a countigous segment of text that consist of one or more EUDs that together form a coherent substructure of the discourse. We also generate one attribute for each one that indicates whether they are nucleus or satellite.
2. **Relations**: RST identifies different types of relations that can exist between textual segments. These relations explain how one segment supports, elaborates, contrasts, or otherwise relates to another. Examples include Cause-Effect, Elaboration, Background, and Contrast.
3. **Hierarchical organization**: Texts are viewed as hierarchically structured, with some segments serving as the nucleus (or central part) of a text, and others as satellites that provide supporting information, elaboration, or contrast to the nucleus.

In the graph-based version of RST, the goal is to eliminate intermediary or terminal nodes. This process transforms the graph so that all remaining nodes are of the same type, specifically EDUs. This ensures that the graph consists only of homogeneous nodes, which is easier to handle in the encoder part of our model. Xu et al. (2020) used Graph Conversion and we follow their setup as well. We have provided some additional examples, using our processed dataset, in Appendix A.1. Lastly, we will simply employ RST as a tool without further developing or expanding it. Therefore, extensive discussion of its details is unnecessary, as those interested can consult existing books that describe this theory.

How

Discourse structure is essential not only for applications like text summarization, but also for sentiment analysis, and question-answering. Despite advancements, the detection of discourse relations remains challenging due to varied lexicalizations and sparse features. The method "Representation Learning for Text-Level Discourse Parsing (DPLP)" (Ji and Eisenstein, 2014) introduces a novel concept that surpasses prior techniques in detecting nuclearity and relation - two essential elements for constructing an RST tree.

The core idea behind DPLP is projecting lexical features into a latent space to facilitate discourse parsing, and to overcome high dimensionality and feature sparsity. This approach uses transition-based structured prediction for discourse parsing. They also use the shift-reduce parsing technique, maintaining a stack and a queue, with the parser deciding whether to shift or reduce elements based on the discourse relation.

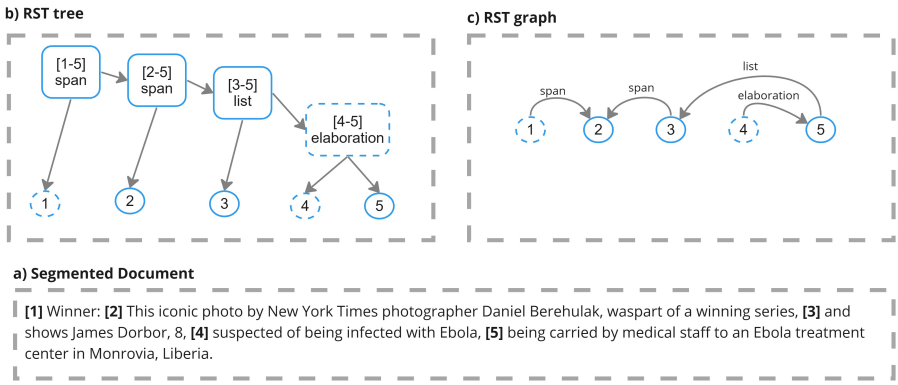


Figure 2.12: concrete example of RST tree conversion to RST graph, adapted from (Xu et al., 2020). Dotted circles are satellite nodes.

After utilizing DPLP for RST parsing, we obtain an RST tree comprising both terminal and non-terminal nodes (a.k.a leaf nodes or EDUs). Each node is classified by the parser as either a nucleus or a satellite. To convert this tree into its graph format, we apply two rules. First, if a node has two nucleus children, we connect the right child to the left child, which will serve as the head. Second, in cases where a node has one nucleus and one satellite, the satellite is connected to the nucleus, with the latter being the head. For instance, as illustrated in Fig 2.12, the terminal node [4-5] has two children, one of which is a nucleus. By applying the second rule, we connect node 4 to node 5. Furthermore, considering the terminal node [3-5], which comprises two children (node 3 and the terminal node [4-5] with node 5 being its head), and following the first rule, node 5 should be connected to node 3. Following this methodology, we can systematically connect the entire RST tree in Figure 2.12 to its graph, as well as the tree in Figure 2.11.

2.4.3 Co-reference Graph

What and Why

Co-reference refers to the linguistic concept where several expressions in a given text refer to the same entity in the world (Sukthanker et al., 2018). It is a vital aspect of understanding natural language because it relates to how different parts of a text connect.

In simple terms, co-reference happens when two or more words or phrases in a sentence or across sentences refer to the same entity. For instance, in the sentence "Reza went to the store. He bought fish," the word "He" is a co-referent of "Reza." They both refer to the same entity.

Co-reference resolution, a task in natural language processing, involves identifying all expressions that refer to the same entity in a text. This is crucial for tasks like information extraction, document summarization, question answering, and machine translation, where understanding the relationships between different parts of the text is important.

Liao et al. (2021) recently showed how critical is co-reference resolution in Dialogue systems. They used one example from the MultiWOZ dataset (Eric et al., 2020), which is a widely used dataset in dialogue management studies, and we have modified their example as Table 2.3. In this example, we observe that "area as the restaurant" has two possible references. The middle of the town, however, is the correct reference and if the model understands the reference differently, it will result in a malfunctioning system. It can result in a factually inconsistent summary in the abstractive text summarization.

Intuitively, if we have one EDU, or utterance in dialogue systems, to which we refer multiple times along the text, that part plays a pivotal role in the overall meaning of the text. Consequently, we can leverage this information in extractive summarization - EDU selection part - as well. Figure 2.13 shows how we can use EDU as the node for our co-reference graph by drawing links of one entity, as an example, between nodes.

Table 2.3: Importance of Co-reference Resolution in Dialogue.

Domain	Dialogue
Restaurant	U: i am looking for a moderate-price british restaurant in south of the town. S: sorry, i am not finding any place [...].would you like to try another area? U: that is fine, how about in middle of the town? S: fitzbillies restaurant serves british food, [...]. U: Good. can you book it please and get me the reference number? S: sure, what day and time would you like and how many people? U: i would like a table for 5 at 11:30 on tuesday [...]
Attraction	S: okay, the booking was successful [...]. anything else i can help you with? U: Im also looking for entertainment in the same area as the restaurant. S: is there any type of attraction you would like me to search? U: why do not you try an architectural attraction.
Taxi	S: all saints church looks good, would you like to head there? U: great, can you find me a taxi to the restaurant first?

How

We will discuss the way that we can find co-references between different entities across different EDUs, then we will see how we can generate a co-reference graph of a document based on this information. As shown in Figure 2.13, in each document we can have multiple entities, like "Pulitzer prizes", to which we can refer across the different parts of the text- different EDUs. Each entity forms a unique cluster with its co-references. Stanford CoreNLP library (Manning et al., 2014) is widely used in preprocessing of data. It is also capable of finding co-reference clusters and Xu et al. (2020) used it for finding co-reference clusters. We do not go into the details of the CoreNLP implementation, however.

Once we have found the co-reference clusters, we can create our co-reference graph (which will be called Coref henceforth) by drawing an edge between all the EDUs that have an entity belonging to the cluster. Both RST and Coref graphs are directed graphs, and Coref graphs have this extra property that if node A is connected to B then B should be connected to A as well, since the co-reference relation is symmetric. Note that nodes in

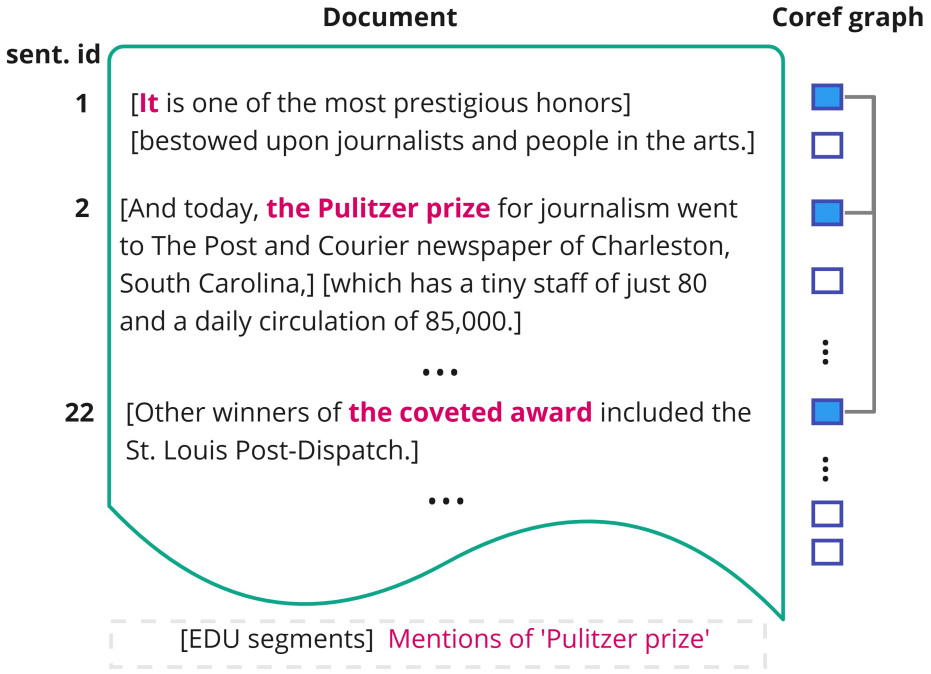


Figure 2.13: Co-reference example, adapted from (Xu et al., 2020).

Coref graph, as well as RST graph, are EDUs.

2.5 Evaluating Summaries

In the field of text summarization, quantifying the quality of a summary presents a unique challenge. This challenge stems from the intricate balance between brevity and completeness, alongside the preservation of the original text's meaning and nuances. Traditional methods of evaluation often fall short in capturing the multifaceted essence of a high-quality summary. Despite these obstacles, the scientific community has leaned heavily on the ROUGE score (Lin, 2004) as a primary tool for assessment. ROUGE, which stands for Recall-Oriented Understudy for Gisting Evaluation, offers a quantitative approach by comparing the overlap of n-grams, word sequences, and word pairs between the generated summaries and a set of reference summaries. However, while ROUGE scores provide a standardized metric for evaluation, they cannot inherently fully encompass the qualitative aspects of summary quality, such as coherence, readability, and the preservation of critical

information.

In subsequent sections, we will delve deeper into the intricacies of the ROUGE score, examining its methodology in greater detail. Following this, we will explore an alternative evaluation approach that promises a more holistic assessment of summary quality.

2.5.1 ROUGE

ROUGE, as its name suggests, is a recall-oriented approach. It measures the amount of information that our system's generated summary can remember (or "recall") from the reference summary. We measure this by counting the ratio of overlapping n-grams to all n-grams in the reference summary. The notation ROUGE-n refers to the ROUGE score with the mentioned ratio calculation by considering n-grams. However, in some recent papers such as (Xu et al., 2020), authors report the F1-score of ROUGE-n when they use ROUGE-n notation without always mentioning that they have this assumption. The exact formula for ROUGE-n Recall, Precision, and F1 are presented by Equations 2.3, 2.4, and 2.5 respectively.

One reason that the F1 score is preferred is that F1 is a geometric mean of Recall and Precision that could be a better representation than the original ROUGE itself. Following Xu et al. (2020), we also use the F1 score to report our ROUGE metric. We also have another variation of ROUGE which is ROUGE-L. Here, 'L' is an abbreviation for Longest Common Subsequence (LCS) and we measure it by dividing the length of LCS by the length of the reference summary.

$$\text{Recall} = \frac{\text{Number of overlapping N-grams}}{\text{Total number of N-grams in the reference summaries}} \tag{2.3}$$

$$\text{Precision} = \frac{\text{Number of overlapping N-grams}}{\text{Total number of N-grams in the generated summary}} \tag{2.4}$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.5}$$

As we can see in the equations above, we calculate the ROUGE solely based on the

overlapping number of n-grams, without considering the subtleties involved in text generation evaluation.

2.5.2 BERTScore

The limitations of ROUGE underscores a broader conundrum in the field of text summarization; the quest for a comprehensive evaluation metric that can holistically capture the quality of a summary. BERTScore (Zhang et al., 2020b) is a novel evaluation metric that employs contextual embeddings from BERT or other transformer-based models to evaluate the semantic similarity between generated text and reference text. Unlike traditional metrics that rely on exact matches of n-grams, BERTScore calculates the similarity of embeddings for each token in the generated text with each token in the reference text, capturing the contextual nuances of language more effectively. It computes precision, recall, and F1 scores by aligning tokens between the candidate and reference texts based on their cosine similarity in the embedding space. Precision is computed by averaging the maximum similarity scores for each token in the candidate text, recall is obtained by averaging the maximum similarity scores for each token in the reference text, and the F1 score is the harmonic mean of these two. This approach enables BERTScore to evaluate text generation tasks with a greater emphasis on semantic meaning, making it particularly beneficial for tasks where lexical diversity and contextual accuracy are crucial.

2.6 Chapter Summary

This chapter provides a comprehensive overview of foundational concepts and advancements in neural networks, language models, and graph neural networks, emphasizing their relevance to abstractive summarization. Having read this chapter, we can have a better understanding of the underlying concepts and techniques used in the following chapters. This chapter begins with an in-depth discussion on neural networks, detailing the single perceptron, activation functions, propagation processes, and regularization techniques. The

chapter then delves into language models, covering tasks, neural models, and the emergence of large language models. It also explores the necessity and techniques of graph neural networks. The related works section reviews various summarization approaches, including extractive, abstractive, and hybrid models. Finally, the chapter addresses evaluation metrics like ROUGE and BERTScore.

Chapter 3

Related Works

This work includes an approach and analysis of multiple aspects, including different datasets, graph types, graph incorporation techniques, hyperparameter design, training methods, evaluation techniques, base language models, and summarization approaches. Our work, however, is mainly in text summarization domain; hence, we briefly review recent related works on this aspect.

Text summarization can be approached in three main ways: Extractive, Abstractive, and Hybrid (El-Kassas et al., 2021). The extractive method tries to find and pick out the most informative phrases and sentences from the text to form a shorter text that keeps the original language with the same wording. It relies on an algorithm to determine which sentences are most important as a key component. In contrast, abstractive summarization generates new phrases and sentences that succinctly convey the core concepts of the original text, akin to how a human might interpret and rewrite the main ideas in their own words. The hybrid approach combines both techniques, leveraging the accuracy of extractive summarization and the creativity and novelty of abstractive summarization to produce summaries that are both accurate and coherent, often resulting in more nuanced and readable outputs. This integrated method aims to harness the strengths of each to optimize the quality and utility of the summarized content. Our approach falls into the hybrid category, incorporating a selection stage followed by an abstraction stage.

In the sections below, we will go into more detail about these three approaches to summarization, using recently proposed models as examples for each of them.

3.1 Extractive Models

The recent paper by Xu et al. (2020) introduced an extractive model named Discourse-Aware Neural Extractive Text Summarization, or DiscoBERT for short, which aims to improve the segmentation of text as well as the selection of text segments. For the segmentation, they use previously introduced ideas to chunk down the document into smaller units than a sentence. This smaller unit of selection was discussed in the previous chapter. They also employ two graph structures for representing the text so that the model can have a better idea of the relationships between units of selection. We elaborate on these units of selection and graph structures in the next chapter.

They have also, implicitly and explicitly, mentioned the reason behind some of their architectural design and hyper-parameter choices. Firstly, they perform an extractive method on CNN/DM dataset (Nallapati et al., 2016), as it is highly suitable for extractive than abstractive summarization. We will elaborate on this in Chapter 4. Secondly, they used a newer unit of selection as it was getting a higher score than the previous approach (refer to Table 2.2). Lastly, they decided to add a graph layer to the BERT model as the model is not able to capture long-range document-level dependencies, thereby lowering the selection accuracy. This is because, as we discussed previously (Section 2.2.3), BERT is trained on MLM and NSP tasks which are token-level and sentence-level tasks respectively, and consequently, BERT will fall short in document-level tasks, like important segments selection.

3.2 Abstractive Models

BART (Bidirectional and Auto-Regressive Transformers) (Lewis et al., 2020) is a transformative model in the field of natural language processing that merges the strengths of BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer) (Radford et al., 2018). Unlike BERT, which is an encoder model that excels at understanding context by analyzing text from both directions, and GPT, which is an autoregressive decoder model that generates coherent text by predicting the next word

in a sequence based on previous words, BART combines both approaches. It uses bidirectional understanding like BERT and autoregressive generation like GPT, allowing it to handle both text comprehension and text generation tasks effectively.This combination enables BART to effectively understand and generate text, making it exceptionally versatile for a wide array of NLP tasks. This is an abstractive model that outperforms recent abstractive and extractive models, like DiscoBERT, by a large margin. BART owes its success, in part, to the larger model trained on a variety of tasks, thanks to the increasing computation resources available.

Zhu et al. (2021) recently used an entity-relation graph to enhance the factual correctness of generated summaries. Their approach is threefold. First, they extract entity relations from the input document and, after a series of transformations, use two GAT layers to obtain embeddings for the nodes. These embeddings are then fused into the decoder block of the transformer to make the output generation more fact-aware. Secondly, they fine-tune a seq2seq model for post-processing, which detects factually incorrect parts and generates the correct versions. Lastly, they propose a simple-to-compute metric for factual correctness that does not rely on neural network models.

See et al. (2017) introduced a different approach for generating the next token in typical seq2seq architecture. They proposed Pointer-Generator Network integrated into a seq2seq model with a pointer mechanism, enabling the model to generate new words and copy words directly from the source text, thus enhancing factual accuracy and handling out-of-vocabulary words. Additionally, a coverage mechanism is employed to reduce redundancy by tracking attention history. Evaluations on the CNN/DM dataset demonstrate that this model significantly improves summary quality, outperforming its baseline models in terms of ROUGE Lin (2004) scores. This evaluation metric will be discussed in the following sections.

Liu et al. (2022) recently attempted to bridge the gap between training time and inference time (or production mode) in abstractive summarization models. Specifically, they

aim to choose a more suitable loss function and training approach to enhance performance on both the training dataset and the test dataset at inference time. During training, models typically rely on one summary and use maximum likelihood estimation to maximize the probability of predicting the tokens in the reference summary. However, during inference, beam search is employed to generate multiple summaries, from which the best output is selected. To incorporate this behavior into the training process, they propose a multi-task loss function that considers both the generation and selection of a good candidate summary.

3.3 Hybrid Models

Liu and Lapata (2019) introduced two extractive and abstractive frameworks which were combined into a stage-wise hybrid model. Their proposed hybrid model, also referred to as BERTSumExtAbs, works on two independently trained stages; extraction with BERT and abstraction with Transformers (Vaswani et al., 2017). Their hybrid model outperforms the abstraction-only model they implemented on all evaluation metrics. This hybrid version, however, is slightly below the extraction module of Liu and Lapata (2019). One reason, as we alluded to earlier, is the type of CNN/DM that makes it suitable for extractive summarization. Another reason might be from gradient disconnection of two modules used in the hybrid model. In chapter 7, we will introduce a technique that can be helpful in addressing this problem.

Edge et al. (2024) recently used graph in retrieval-augmented generation (RAG) systems for query-focused summarization tasks. They first split the source documents into text chunks for processing. Instead of using the commonly employed knowledge triples (subject, predicate, object), they use (entity node, relationship edge, claim covariate) as their graph elements. The extraction of each element from the source chunks and the element-level summary generation operations are both performed using LLM. Different community detection methods can then be applied to partition the graph into topics (or communities). An LLM is then used to generate summaries for each community to facilitate global ques-

tions. These steps are carried out during the indexing phase. During the query phase, community answers are generated from the community summaries to help produce the final global summary, all performed by an LLM (Edge et al., 2024).

Wang et al. (2017) have also approached the summarization of long documents in a hybrid manner. They used a typical encoder-decoder architecture for their abstraction module but handled sentence selection in their extraction module in a novel way. In the sentence extraction phase, the model uses a graph-based approach to identify key sentences from a document. The text is represented as a complete graph where each node corresponds to a sentence, and edges between nodes are weighted by a combined similarity measure incorporating sentence vector similarity and Levenshtein distance (Levenshtein, 1966). The PageRank (Page et al., 1999) algorithm is applied to rank sentences based on their importance, with top-ranked sentences being selected for extraction. The similarity measure incorporates both semantic and literal similarities, using cosine similarity of sentence vectors and Levenshtein distance, respectively. This combined measure ensures a more accurate representation of sentence importance. The final extraction retains the original order of sentences while selecting the most relevant ones.

3.4 Chapter Summary

This chapter reviews recent related works in text summarization, categorizing them into extractive, abstractive, and hybrid models. Extractive models, like DiscoBERT, focus on selecting important text segments, while abstractive models, such as BART and FASum, generate new phrases to convey core concepts. Hybrid models, including BERTSumExtAbs and graph-based approaches, combine both techniques for improved accuracy and coherence. The chapter also highlights innovations like the Pointer-Generator Network for handling out-of-vocabulary words and BRIO for enhancing training-inference consistency. Each approach is discussed with examples of recent models, providing a overview of the text summarization domain.

Chapter 4

Dataset Preparation

The focus of this work is graph-based summarization over Elementary Discourse Units (EDUs). Given this requirement, Xu et al. (2020) has already preprocessed CNN/DM dataset (Nallapati et al., 2016) by segementing documents into EDUs and generating two types of graphs used for the text summarization purpose. We undertook a costly process of dataset processing to enhance the quality of our analysis, aiming to reduce bias and increase the robustness of our summarization models.

The CNN/DM dataset's important content distribution introduces a bias in our study that needs to be addressed. Gupta et al. (2021) demonstrated that the CNN/DM dataset is highly suitable for extractive summarization, with the most important parts typically located at the beginning of the documents. Furthermore, Xu et al. (2020) conducted a numerical analysis of this dataset. Their findings indicate that the Lead3 model, which simply selects the first three sentences of a document as its summary, performs nearly as well as more sophisticated neural network approaches, despite its naive simplicity.

Measuring the model's performance on other related but distinct datasets can test the robustness of our approach. Considering the wide variety of summarization datasets available, we chose XSum (Narayan et al., 2018), one of the most widely used datasets in the text summarization task, alongside CNN/DM and NYT (Sandhaus, 2008). Unlike CNN/DM, XSum summaries are highly abstractive (paraphrased) and often significantly shorter. These characteristics pose multiple challenges to our model and the EDU selection approach, which we will discuss in detail in Section 7.1.

There were additional dataset candidates to consider as well. For example, the NYT dataset is one of the most frequently used. However, this dataset is not open source, and using it would have prevented us from publishing our processed version to the community. Other datasets, such as DUC (Harman and Over, 2002) and Gigaword (Graff et al., 2007), were not suitable due to accessibility issues, size constraints, and language limitations, making them less effective for our study. Nevertheless, following our clear instructions, another dataset could be processed for different purposes. Table 4.1 provides an overview of these datasets along with their features.

Table 4.1: Comparison of Text Summarization Datasets.

Feature	**DUC**	**CNNDM**	**Gigaword**	**NYT**	**XSum**
Size ≈	Variable by year	0.3M articles	10M documents	1.8M articles	0.2M articles
Source	News articles from various sources	CNN and Daily Mail news stories	News articles from agencies like NYT, AP, Xinhua	New York Times articles	BBC article summaries
Type	News articles with human-written summaries	News articles with bullet-point summaries	News articles with headlines as summaries	Full-text articles with abstracts	News articles with one-sentence summaries
Use case	Multi-document summarization	Single-document summarization	Sentence-level extractive summarization	Multi & single document summarization	Single-document abstractive summarization
Access	Restricted, application required	Publicly available for research	Licensed access for academic research	Licensed access, primarily academic	Publicly available for research
Language	Mostly English	English	Mostly English	English	English

In the following sections, we will discuss our pipeline for processing XSum and will briefly described the format of processed CNN/DM.

4.1 Processed CNN/DM

By using the aforementioned tools in this chapter, Xu et al. (2020) processed each document of CNN/DM and provided the processed version in multiple compressed files (.pt

files) on their repository[1]. In the following table, we examine the different attributes of their processed files.

Table 4.2: Description of Keys in the Processed CNN/DM

Key	**Description**	**Key**	**Description**
src	Tokenized input	labels	Labels of the sentences
segs	Segmentation of the src into sentences	clss	The index of CLS token in the src
sent_txt	The text of the sentences	disco_txt	The text of the EDUs
tgt_list_str	Reference summary	d_label	Label of EDUs
d_span	Start & end ids of EDUs in the src	d_coref	Coref graph
d_graph	RST graph	disco_dep	Undescribed
doc_id	Document id		

In our proposed model, we need to use attributes "d_labels", "disco_txt", "d_graph", "d_coref", and "tgt_lst_str" which are discourse(EDU) labels, discourse text, discourse RST graph, discourse Co-reference graph, and target sentences (oracle summary) respectively. Other attributes are related to sentences (as the atomic unit of selection) and doc-id is the original document ID. We, however, have not found any use case for "disco_dep" in the original paper (Xu et al., 2020) and we have not found any explanation about this attribute.

4.2 Processed XSum

In this section, we will shed light on the steps that we carried out to process XSum for our purpose by using the concepts and the tools we have covered in the previous sections. Finally, we report the structure and statistics of our final processed XSum dataset. We shared the processed dataset, our code, instructions, and extensive details and guidelines with the community on our GitHub repository.[2]

[1] https://utexas.app.box.com/v/DiscoBERT-ACL2020.
[2] https://github.com/Reza-Ardestani/XSum.

4.2.1 Pipeline Overview

Using the pipeline depicted in Fig 4.1, our objective is to generate EDUs, assign labels to these EDUs, and construct the RST graph for each raw XSum document. There are some byproducts in this process, namely nuclearity or RST tree, that one can use in a different study.

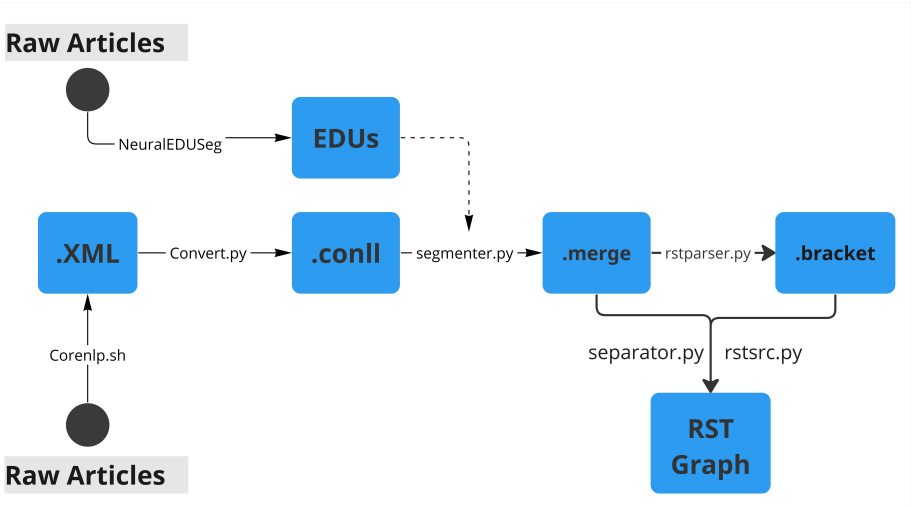


Figure 4.1: XSum Data Processing Pipeline.

To begin with, we start from the lower left of Fig 4.1 which uses Stanford CoreNLP[1] to transform raw documents into .XML files which contains nine attributes for each token in the raw document. Attributes are: sentence index, token index (within a sentence), token, lemma, POS (Part Of Speech tags), dependency label, dependency head, NER (Named Entity Recognition), and partial constituent parse.

At the second stage, we generate .conll files that are a simplified format of .XML files. They contain the same information as .XML files, however. Then, from .conll files we generate .merge files that has the same information as .conll file with another attribute, that is the token's EDU index. In fact, EDU segmentation is done at this stage. Moving forward we generate .bracket files that contain RST tree of documents. Lastly, from both .merge and .bracket files we generate RST graphs of the documents. Now, we have both EDUs and

[1] https://github.com/stanfordnlp/CoreNLP.

RST graphs. To generate the label of EDUs, we use the previously discussed algorithm.

There is also one other arrow joining the .conll to .merge phase. This is because the aforementioned generated EDUs, by DPLP, are not highly accurate. So, we use another repository to generate the EDU index of tokens and use this result to update the original .merge files. Now, the processed documents are ready to be used in our graph-based approach.

There are some important general notes to consider:

- There are some XSum articles that have missing "document" fields and I have removed them. The id of these documents can be found on our XSum repository.[1]
- In case you want to process long documents, You need to change -mx2g to a higher number (like -mx7g). Since I have truncated the documents to have less than 768 tokens, I did not need to change the 2 Gbytes java heapsize to a higher number.
- Sometimes corenlp.sh does not generate .XML file for some documents. In this case you just need to run that one more time on those documents that were left unprocessed. Running one more time usually fixes the problem. Otherwise, try more times.
- Without losing the generality of the problem, I have removed '[' and ']' characters from the documents since corenlp.sh fails to properly generate .XML files of documents containing those two characters.

For each part of the pipeline, we have added one sample file in Appendix A.1 to illustrate their formats and connections.

4.2.2 Server Configurations

Generating .XML files is time-consuming[2]. Therefore, enlisting the help of multiple fast servers was inevitable. We have used one of the servers of our NLP lab at the University

[1] https://github.com/Reza-Ardestani/XSum.

[2] by using one of our lab's servers it will take 29 days to process the whole XSum dataset.

of Lethbridge with the configuration summarized in Table 4.3.

Table 4.3: Local Server Configuration For Dataset Generation.

Component	Specification
GPU Type	12G NVIDIA TITAN
GPU Driver Version	XP530.41.03
RAM	64G
CUDA Version	12.1
OS	UBUNTU 16.04

Processing 10K documents for generating .XML files takes two and a half days with this local server. We batched the XSum documents into 10K-size zipped files and then distributed the processing. The total number of XSum documents is 226k; hence, we had 23 batches of documents to process. Alongside our server, we used five servers of Compute Canada[1] for processing. Unlike our local server, we had queue time each time we ran a job, ranging from one hour to three days. The script for running the code on these remote servers, as well as other supplementary information, are provided on our XSum repository.[2]

4.2.3 Generating Elementary Discourse Units

The pipeline generally relies on DPLP library and this library can segment documents into EDUs as well. DPLP, however, does not perform well on this part and it mostly segment documents into sentences. As a result, we used a NeuralEDUseg (Wang et al., 2018) which has performed very well on this task. Their code, however, is not easily executable since many tools and libraries that they rely on are either unavailable or have version conflicts. Therefore, we needed to modify their code[3]. Li et al. (2020) also modified NeuralEDUseg since they are using it, partly, in their approach. Their code, which is available on their GitHub, can be executed with minor modifications, like changing paths.

Once we generated the .conll files and also we segmented the documents with NeuralEDUseg, we append the EDU number for each token in .conll files. There is a minor

[1] https://docs.alliancecan.ca/wiki.

[2] https://github.com/Reza-Ardestani/XSum/tree/main/DPLP.

[3] Availabel at https://github.com/Reza-Ardestani/XSum/tree/main/NeuralEDUsegmentor.

mismatch here, however. The tokens that we feed into DPLP to generate .XML then .conll will not remain distinguishable. When we have a document such as “He said: I’m not going home.” CoreNLP tokenizes them differently each time that we pass such sentences to the pipeline. The part “said: I’m” one time, for example, will be tokenized by CoreNLP into [“said:”, “I”, “m”] and sometimes into different forms like: [“said”, “I”, ”m”], [“said”, “Im”], etc. So, for finding the EDU to which each token belongs, we have two options. We can use Minimum Edit Distance, or before passing raw articles into CoreNLP, we can add some special tokens, like EDUSEPARATOR, based on the segmented documents that we have already received from NeuralEDUseg code. We implemented both approaches for verifying their correctness, but used the later for its faster runtime.

4.2.4 Generating graphs of Rhetorical Structure Theory

The process for generating RST graphs is documented on our XSum GitHub repository. DPLP library indexes nodes from 1; hence, We need to shift the indices by 1 for them to be indexed from zero. One more special case is that if the document consists of only one EDU, then we are not going to have any graph or tree for that document. We need to handle this case in our summarization process. Table 4.4 shows one of these documents.

Table 4.4: Sample Document With Empty RST Graph.

Key	Value
edu	[“friends, collaborators and fans have been paying tribute on their social media accounts.”]
RST graph	[]
labels	[]
summary	james horner, the hollywood composer who wrote the oscar-winning score for titanic, has died in a california plane crash aged 61.

This sample is from test articles of XSum dataset, which also exemplifies an occasional noise in the original dataset as this summary is unrelated to the document.

4.2.5 Greedy Labeling

Both the abstraction and extraction part of our proposed model are supervised, meaning that we need to have the data labeled by humans in order to train the model. For the extraction part, particularly, we need oracle[1] labels for EDUs as to whether they are important or not. Then we can use these oracle labels (referred to as y) to compare to the predicted labels (referred to as $\hat{y}$).

We discussed about using the ROUGE score as our evaluation metrics, despite its limitation, as the text summarization metric and selected EDUs are expected to increase this score. If there are N EDUs in the document, the total number of choices are $\sum_{i=0}^{N} \binom{N}{i}$ which is 2^n. In this case, it is not feasible to find the optima by brute-force approach.

One way that Xu et al. (2020) has shown to be effective is by selecting using a greedy approach. In this approach, the greedy selection function takes a list of EDUs (or sentences) from the document, a list of EDUs (or sentences) from the oracle summary, and the desired size of the summary.[2] Then, it uses a helper function used to clean EDUs by removing non-alphanumeric characters. The function first preprocesses the input EDUs by cleaning them and splitting them into words.

For each EDU in the document, it calculates the 1-gram and 2-gram separately from other EDUs. Note that N-gram and therefore ROUGE are not additive. This means:

$$ROUGE(\text{EDU}_i + \text{EDU}_j) \neq ROUGE(\text{EDU}_i) + ROUGE(\text{EDU}_j).$$

Consequently, this difference makes our ROUGE score different from the original ROUGE value.[3] As an example, if we consider oracle summary: "Winner was an iconic photo", EDU_i: "Winner was", and EDU_j: "an illustration" then for ROUGE-2 we have:

[1] a.k.a gold, ground truth, reference, human-generated, or manual labels.

[2] The greedy method requires us to segment the document and summary, and it works the same whether we divide the text into sentences or EDUs. The code still correctly select the best segments in a greedy approach.

[3] Without this minor difference, we needed to decide about the order of selected EDUs which would have increased the total number of possibilities from 2^n to $\sum_{i=1}^{n} n!$ for the brute-force and would have added an overhead for our greedy method.

Bigrams:

Gold: (“Winner, was”), (“was, an”), (“an, iconic”), (“iconic, photo”)

EDU_i: (“Winner, was”)

EDU_j: (“an, illustration”)

ROUGE-2 Recalls:

$$ROUGE(\text{EDU}_i) = \frac{1}{4} = 0.25, \quad ROUGE(\text{EDU}_j) = \frac{0}{4} = 0.00$$

$$ROUGE(\text{EDU}_i) + ROUGE(\text{EDU}_j) = 0.25$$

$$ROUGE(\text{EDU}_i + \text{EDU}_j) = \frac{2}{4} = 0.50$$

$$0.50 \neq 0.25$$

Finally, we iteratively select sentences that, when added to the current selection, maximally increase the overall ROUGE score of the selection with respect to the oracle summary. This is done up to the desired summary size. The selection process is greedy; it picks the next best sentence based on the current selection, without reconsidering previous choices. Then, the function returns the indices of the selected sentences. The code that we wrote for the selection, adopted from (Xu et al., 2020), is available at our GitHub repository[1]

If you run the greedy labeling method from the DiscoBERT repository on their provided processed CNN/DM, you do not get the exact labels as they have provided. There is an approximately 96 percent overlap between their labels and predicted labels by us using their code. The difference might be because of some other code that they have not provided.

Secondly, if the summaries are too abstract, like some of the gold summaries of the XSum dataset, then our ROUGE-based greedy labeling approach falls short. While ROUGE measures lexical similarity, other approaches, like BERTScore, measure semantic similarity

[1] https://github.com/Reza-Ardestani/XSum/tree/main/labeling.

that will solve the issue of having no EDU selected as important. However, the effectiveness of BERTScore cannot be extended to other issues, such as the overfitting problem discussed in the next chapter.

Table 4.5: Sample Processed Document With No Important EDU Selected.

Key	Value
edu	[“7 august 2015 last updated at 23:21 bst”, “he decided to head to calais on a fact - finding mission.”, “newsnight’s james clayton and jack garland followed him around for a day.”]
RST graph	[[0, 1, ‘span’], [0, 2, ‘ROOT’]]
labels	[]
summary	ukip mep mike hookem believes british and french authorities aren’t doing nearly enough to stop migrants from coming to the uk.

Table 4.5 shows one example from our processed XSum in which we have no EDU selected as important. It also reveals several factors. Despite being widely used, XSum dataset does not have high-quality records all the time. There are some documents that have an irrelevant summary to the document. In some cases, that the summary is relevant, due to its extreme abstraction, there is no common n-gram between the original document and its summary[1]. In this case, our greedy labeling does not select any EDU as important. The RST graph and EDU list of Table 4.5, however, are generated as expected and correctly.

4.2.6 Final dataset

In this part, we will present the structure and statistics of the processed XSum. We followed the original XSum train, validation, and test split[2] in which we have 204045, 11332, and 11334 articles respectively. The reason for adhering to the original split ratio and document order is that researchers typically maintain these parameters for consistency and comparability of results. The processed articles, all available on our processed XSum GitHub, are saved in JSON format where the train dataset is divided into 10 JSON files for

[1]Note that in our labeling code, we have: _get_word_ngrams(n, sentences, **rm_stop_unigram=True)** indicating that we do not conseider stop words when we are calculating ROUGE scores in the greedy method

[2]We downloaded the XSum dataset from Python standard “datasets” library.

better loading. Table 4.4 shows a sample of the records in our processed XSum dataset. We have summarized some informative aspects of our processed dataset in Table 4.6. More details, such as the frequency of RST graph edge types, are available on our GitHub page.

Table 4.6: Statistics of the Processed XSum.

Dataset	Average		Empty			Total Records
	EDUs	Summary	RST Graph	Oracle Label	Both	
Train	52.35	21.10	121	1944	102	204017
Test	41.85	21.10	7	85	5	11333
Valid	41.39	21.13	10	96	8	11327

Note: Average column shows average number of EDUs in articles and average number of words in summaries of articles. Empty indicates how many documents have empty RST graph, oracle labels, or both.

4.3 Chapter Summary

In this chapter, we began by introducing various benchmark datasets for the text summarization task, justifying the selection of XSum as a secondary dataset for evaluating our proposed model. We covered two essential reason for dataset generating another dataset, and discussed each part of the from the dataset preparation pipeline, and showcased the processed XSum dataset's statistics. Throughout, we highlighted the challenges and discrepancies encountered, providing future researchers with a realistic understanding of the time, resources, and benefits involved in generating a new dataset.

Chapter 5

Proposed Model

In this chapter, we will provide an overview of our proposed model before diving into the specifics of its two stages. After discussing the nuances of training, fine-tuning, and inference, we will present the experimental environment to facilitate replication.

5.1 Model Overview

Our goal is to develop a hybrid model that operates in stages, utilizing an encoder for extraction and a decoder for coherent narration of the extracted content. The encoder segment employs various encoding layers to assess EDUs, classifying them as either critical or trivial. We fine-tune this initial stage using oracle labels in a supervised fashion, selecting the iteration that optimizes the F1 score for classification. Following this, we merge the identified EDUs to serve as fine-tuning data for the next phase, our abstractive sequence-to-sequence (Seq2Seq) model. This model is adept at refining essential EDUs into a summary that aligns with the quality of human-generated summaries, eliminating superfluous details. Finally, the model's performance is evaluated against oracle summaries to calculate the loss value, which guides the selection of the optimal abstractive model, also referred to as the decoder section. Figure 5.1 showcases the integration of these phases, which in total, form our hybrid model.

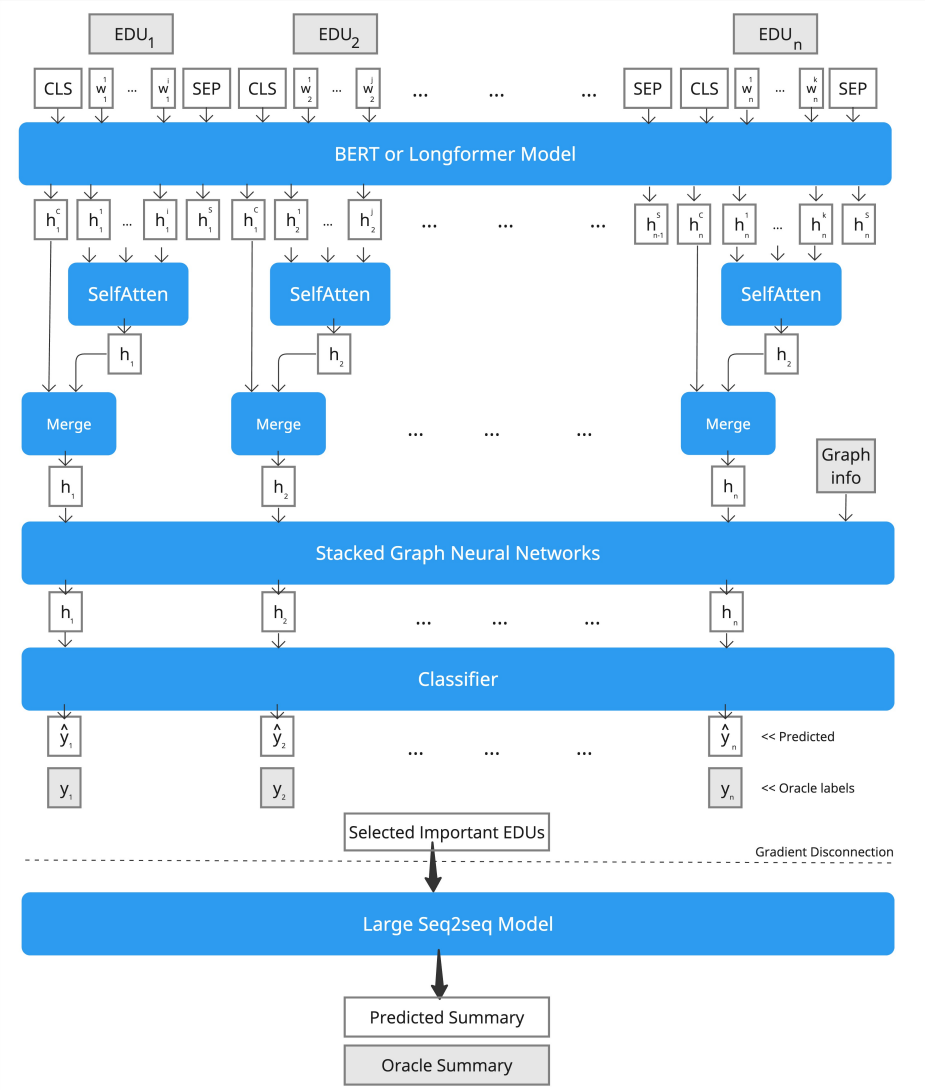


Figure 5.1: Our proposed model overview.

5.2 Encoder

In this section, we will delve into the six essential steps that constitute our encoder module, providing a detailed examination of each. Initially, we append a CLS (classification) special token to the beginning of each EDU to assist the model in recognizing each EDU as a distinct entity for classification purposes. Following this, we add a SEP (separator) special token to delineate the boundaries between consecutive EDUs within the input sequence. We followed Xu et al. (2020) for this input format.

Secondly, we leverage a pre-trained language model to produce hidden representations

for each token in our input sequence, including the special tokens as well. For this component, we had several options. We chose Longformer as the first encoder language model due to its intrinsic ability to process lengthy inputs, supporting sequences up to 4,096 tokens through a combination of sliding window (local) attention and global attention mechanisms while BERT's maximum input length is 512 tokens. This choice enabled our encoder to process CNN/DM input documents, some of which exceed the 512-token maximum length, without significantly increasing the computational cost. Moreover, we test our approach using a second encoder language model, BERT, to assess our approach further.

Now that the model has generated a hidden representation for each token we need one single representation for each EDU span, that is from varying lengths. For this purpose, we followed (Xu et al., 2020) and used a modified attention mechanism (Vaswani et al., 2017) to aggregate contextualized token embeddings into a single vector. This module, SelfAttentiveSpanExtractor that is referred to as SelfAtten in Figure 5.1, can be formalized in several steps:

1. **Linear Transformation and Non-linearity**.

   Given a span embedding matrix $X \in \mathbb{R}^{n \times d}$, where n is the span length and d is the dimensionality of the token embeddings, the module first applies a linear transformation followed by a Rectified Linear Units (ReLU), (Nair and Hinton, 2010), activation function:

   $$H = \text{ReLU}(XW_1 + b_1)$$

   where $W_1 \in \mathbb{R}^{d \times d'}$ is a weight matrix, b_1 is a bias vector, and d' is the hidden dimension. The ReLU function is defined as $\text{ReLU}(z) = \max(0, z)$, introducing non-linearity into the model and enabling it to learn complex patterns.

2. **Attention Score Computation.**

   The module computes an attention score for each token within the span by applying another linear transformation to H:

$$A = HW_2 + b_2$$

where $W_2 \in \mathbb{R}^{d' \times 1}$ is a weight matrix and b_2 is a bias term. The result $A \in \mathbb{R}^{n \times 1}$ is a column vector where each element corresponds to the unnormalized attention score for each token.

3. **Attention Weight Normalization.**

   To ensure the attention scores are comparable across different spans and to interpret them as probabilities, we apply the softmax function:

$$\alpha_i = \text{softmax}(A) = \frac{e^{A_i}}{\sum_{j=1}^{n} e^{A_j}}$$

   for each attention score A_i in A. This normalization step produces a set of attention weights $\alpha \in \mathbb{R}^{n \times 1}$, where each weight is between 0 and 1, and the sum of all weights in a span is 1.

4. **Weighted Sum and Span Representation**.

   Finally, the module computes the span representation as the weighted sum of the original token embeddings, using the attention weights:

$$S = \sum_{i=1}^{n} \alpha_i X_i$$

   where $S \in \mathbb{R}^d$ is the final span representation, and X_i is the i-th token embedding in the span. This step effectively allows the model to emphasize more relevant tokens and diminish the contribution of less relevant ones, producing a contextually rich and fixed-length representation of the span.

The fourth step involves combining the CLS representation of each EDU with its corresponding span representation through a simple linear layer. Following this, we have a

stacked GNN component to cultivate a more nuanced representation of the EDUs. Specifically, we used three GATs that uses a similar attention mechanism that we described in SelfAttentiveSpanExtractor and works in the manner that we demonstrated in Equation 2.1. Lastly, we pass the representations to a classifier that predicts the class of each EDU.

Other than experiments using graphs with the GAT architecture, we have two other types of experiments. In the experiments named "without GAT", we deactivated the GAT layers while keeping the classifier layer in place. Secondly, we experimented with different architecture for encoding graph information than GAT. We used MLP that takes one-hot vector representation of graph information, illustrated in Figure 5.2. As we are using this one-hot representations at the final layers of our model, the computational cost of it is negligible. In this figure, CLS and EDU embeddings are generated the same way as Figure 5.1 and here we do not repeat the same parts of the Encoder module in the illustration. To find optimal hyperparameters of the Encoder, we ran extensive experiments on different aspects, including architecture. Later, we discovered that using one classification tower and passing both *CLS and EDU* embeddings and graph information yields in better results. Coref share is the percentage of input edges for each EDU, which is only used for CNN/DM as we do not have Coref graph information for XSum dataset.

To elaborate more, we demonstrate how the input vector for an EDU is generated, which is then passed to the MLP layer for predicting its importance. Figure 5.3 illustrates the RST and Coref relationships between EDU*i* and its neighboring nodes. There are 43 different RST relation types, including span, root, elaboration, comparison, and result, among others. Each EDU may have multiple incoming and outgoing RST edges. We represent this information using one-hot vector encoding, where a '1' in the vector indicates the presence of an incoming or outgoing RST edge with that specific relation. The Coref share value represents the percentage of incoming Coref edges for each EDU. For instance, if there are a total of 10 Coref edges in the document and EDU*i* receives two of them, its Coref share value would be 0.2. As the majority of documents have fewer than 169 EDUs, we chose

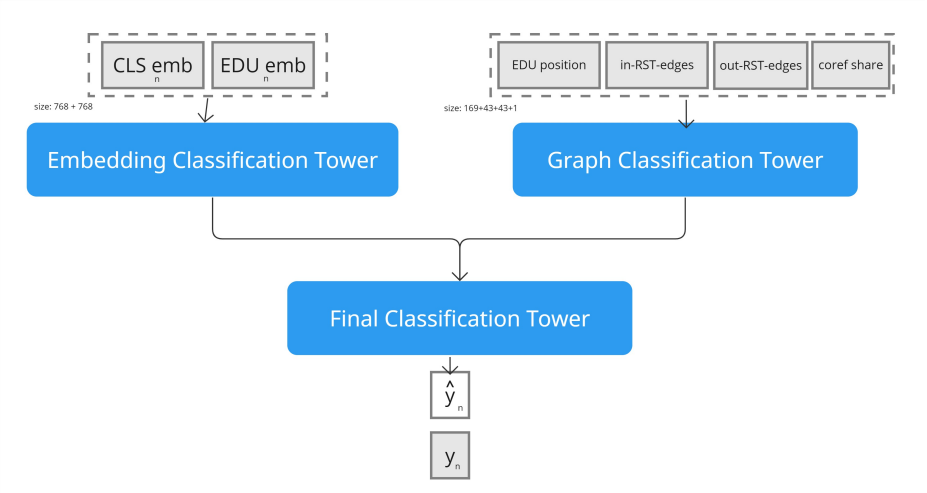


Figure 5.2: Using MLP in incorporating graph information. CLS and EDU embeddings are generated by BERT or Longformer model as illustrated in Fig 5.1.

169 as the size of the EDU position vector, which makes the final dimension of the vector 2^8 (or $256 = 169 + 43 + 43 + 1$).

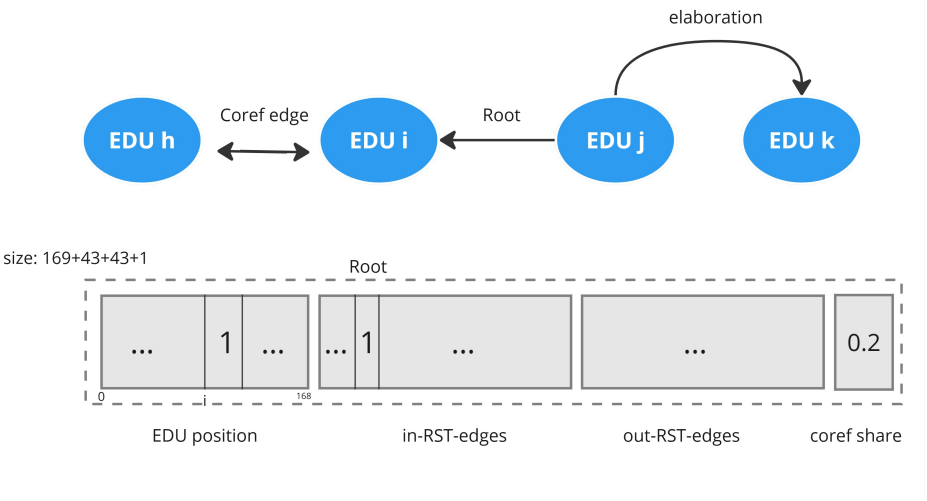


Figure 5.3: MLP input format of EDU_i.

5.3 Decoder

We had multiple options, such as BART or Pegasus (Zhang et al., 2020a), for our Seq2Seq model. Pegasus performs slightly better in R-2 and R-L metrics than BART. How-

ever, we chose BART to fine-tune, due to its fewer number of parameters. We downloaded the BART-base from HuggingFace repository[1] for CNN/DM and BART-Large[2] for XSum dataset. Then, fine-tuned them on our extracted EDUs from the previous stage almost based on the hyperparameters that they have given on these repositories. As XSum dataset has fewer number of articles and significantly shorter summaries, we were able to use the large version of BART in its experiments.

5.4 Fine-tuning and Inference

During the model fine-tuning process, several key considerations were addressed. Notably, over 90% of the EDUs were unimportant and labeled as zero, leading the model to frequently predict everything as zero. To counteract this imbalance, we increased the weight assigned to positive labels. Initially following the approach outlined in Xu et al. (2020), we utilized Binary Cross Entropy (BCE) loss for our binary classification task. However, we later redefined our task within the framework of logistic regression and opted for MSE loss, which significantly improved our selection F1 score from 35% to 39%, approximately. This improvement was also aided by the incorporation of dropout in the classification layer, the final segment of our encoder. This regularization technique effectively mitigated overfitting on the training set, enhanced model generalization, and thereby boosted validation dataset performance.

Another pivotal factor in our model's performance was the implementation of a learning rate scheduler. This tool was instrumental in gradually reducing the step size as the model approached the later stages of fine-tuning, which are presumptively closer to the optimal solution. Without this adjustment, the constant large steps from the initial fine-tuning phase could potentially overshoot the optimum, preventing convergence.

Our encoder model, as illustrated in Figure 5.1, comprises multiple components, including pre-trained language models. However, other components are not pre-trained. There-

[1] https://huggingface.co/ainize/bart-base-cnn.
[2] https://huggingface.co/facebook/bart-large-xsum.

fore, we first freeze the weights of our pre-trained model and train the encoder network for 12 epochs. Then, we unfreeze and fine-tune the entire encoder network for a few epochs. This approach allows the layers after the language model, which are initially randomly initialized, more time to adjust themselves.

For the encoder, no hyperparameters required adjustment at inference time. However, the decoder's inference phase presented multiple adjustable parameters. We adhered to the length penalty as per the model's documentation on HuggingFace but customized the maximum length, minimum length, and number of beams.

5.5 Experimental Setup

Our primary computational resources were Cedar and Narval servers from Compute Canada[1], which are equipped with V100, and A100 GPUs each having 32G, and 40G of memory, respectively. To accelerate the fine-tuning process, we employed the Distributed Data Parallel (DDP) feature from PyTorch[2] following the configuration guidelines provided on the Compute Canada data parallelization webpage[3]. This setup allowed us to utilize 4 GPUs simultaneously, significantly enhancing our fine-tuning efficiency. Full details, scripts, code, and utilities of our research are available on our GitHub repository.[4]

5.6 Chapter Summary

This chapter presented a hybrid model for text summarization, featuring an encoder that identified essential EDUs and a decoder that generated coherent summaries. The encoder used Longformer and BERT for token representation and a self-attentive span extractor to generate EDU representations, incorporating graph information through GAT or MLP. Fine-tuning involved handling class imbalance with weighted labels and MSE loss, regularizing with dropout, and optimizing with a learning rate scheduler. The decoder, based

[1] https://docs.alliancecan.ca/.
[2] https://pytorch.org/docs/stable/generated/torch.nn.parallel.DistributedDataParallel.html.
[3] https://docs.alliancecan.ca/wiki/PyTorch.
[4] https://github.com/Reza-Ardestani/GraphSummarizer.

on BART, refined extracted EDUs into summaries. Training and inference optimizations included freezing pre-trained model weights and adjusting beam search parameters. Experiments were conducted using high-performance GPUs and distributed data parallelism, which lowered the run time of our numerous experiments. We have provided an approximation for the run times on our GitHub repository.[1]

[1] https://github.com/Reza-Ardestani/GraphSummarizer.

Chapter 6

Evaluation

In this chapter, we will present the results of our experiments and evaluate the effectiveness of our proposed approach. We will assess the impact of various factors, including different base models, graph types, graph incorporation architectures, and datasets, on the performance of our encoder models. The results will be demonstrated through tables that showcase the performance on the CNN/DM and XSum datasets. We will also compare our results with previously published models as our baseline (Lewis et al., 2020; Xu et al., 2020; Liu and Lapata, 2019).

6.1 Results

In our study, we conducted a series of experiments from which we analyzed the findings. As previously mentioned, our analysis of our proposed approach is manifold. We evaluated the impact of various factors, including different base models, graph types, graph incorporation architectures, and datasets. Table 6.1 and Table 6.2 demonstrate the performance of our different encoder models on CNN/DM and XSum datasets. Our primary criterion for evaluating the optimal encoder module was based on the F1 selection score. Although Longformer-based models generally achieve lower F1 score than their BERT-based counterparts, they have a window size that is almost three times longer.

We use our fine-tuned encoder modules to predict important EDUs which will be fed into the final stage, which uses BART model. Upon applying the decoder module over the concatenated EDUs, we obtain the final summary generated by our hybrid approach. We

Table 6.1: Test Results of the Encoder Models on CNN/DM Dataset.

Model	Precision	Recall	F1 Score
Longformer without GAT	28.98	56.94	38.41
Longformer with GAT for RST	28.75	54.58	37.66
Longformer with GAT for Coref	26.57	57.26	36.30
Longformer with GAT for RST and Coref	27.62	57.94	37.40
Longformer with MLP for RST and Coref	29.81	54.94	**38.65**
BERT without GAT	28.61	65.00	39.74
BERT with GAT for RST	29.63	61.64	40.02
BERT with GAT for Coref	29.63	61.84	40.06
BERT with GAT for RST and Coref	29.74	61.89	**40.18**
BERT with MLP for RST and Coref	29.64	62.17	40.15

Table 6.2: Test Results of the Encoder Models on XSum Dataset.

Model	Precision	Recall	F1 Score
Longformer without GAT	30.71	50.04	38.06
Longformer with GAT for RST	30.89	48.06	37.61
Longformer with MLP for RST	30.86	49.74	**38.09**
BERT without GAT	31.11	51.48	38.78
BERT with GAT for RST	31.29	50.69	38.70
BERT with MLP for RST	32.52	48.85	**39.05**

calculated the ROUGE scores using compare_mt library[1]. Table 6.3 and Table 6.4 depict the evaluation result of our generated summaries which can be compared with previous models by other authors.

Our best model on CNN/DM dataset is compared with our baseline models in Table 6.5. The term "Lead3" refers to the first three sentences of the input document. Remarkably, by simply selecting Lead3, the results approach the performance of our baseline models. This observation underscores the extractive nature of the CNN/DM dataset. In this table, we present three baseline models that closely resemble our proposed model, spanning three different categories of summarization systems.

Our best fine-tuned model on the XSum dataset as well as experiments on oracle EDUs are presented in Table 6.6. Prior to committing extensive computational resources, we con-

[1]https://github.com/neulab/compare-mt.

Table 6.3: Test Results of Encoder-Decoder Models on CNN/DM Dataset.

Model	R-1	R-2	R-L
Longformer without GAT	43.60 ± 0.23	20.77 ± 0.25	40.60 ± 0.22
Longformer with GAT for RST	43.01 ± 0.23	20.35 ± 0.25	40.00 ± 0.21
Longformer with GAT for Coref	43.04 ± 0.22	20.32 ± 0.25	39.98 ± 0.24
Longformer with GAT for RST and Coref	43.30 ± 0.23	20.55 ± 0.23	40.30 ± 0.23
Longformer with MLP for RST and Coref	**43.73** ± 0.22	**20.91** ± 0.26	**40.72** ± 0.24
BERT without GAT	43.43 ± 0.24	20.62 ± 0.24	40.40 ± 0.23
BERT with GAT for RST	43.37 ± 0.22	20.65 ± 0.25	40.34 ± 0.23
BERT with GAT for Coref	43.40 ± 0.23	20.64 ± 0.24	40.39 ± 0.21
BERT with GAT for RST and Coref	43.34 ± 0.23	20.64 ± 0.25	40.34 ± 0.24
BERT with MLP for RST and Coref	43.42 ± 0.23	20.70 ± 0.25	40.34 ± 0.23

Average ROUGE scores are obtained by 1000 replicates and ± is followed by an estimated margin of error under 95% confidence (Koehn, 2004). The margin of error for ROUGE scores is calculated as the average difference between the lower and upper bounds of the scores.

Table 6.4: Test Results of Encoder-Decoder Models on XSum Dataset.

Model	R-1	R-2	R-L
Longformer without GAT	36.56 ± 0.25	14.28 ± 0.23	28.29 ± 0.23
Longformer with GAT for RST	36.24 ± 0.26	14.14 ± 0.23	28.14 ± 0.24
Longformer with MLP for RST	**36.82** ± 0.27	**14.66** ± 0.24	**28.67** ± 0.26
BERT without GAT	36.41 ± 0.28	14.23 ± 0.25	28.37 ± 0.27
BERT with GAT for RST	35.33 ± 0.25	13.30 ± 0.23	27.45 ± 0.25
BERT with MLP for RST	35.97 ± 0.25	13.95 ± 0.24	27.90 ± 0.26

Average ROUGE scores are obtained by 1000 replicates and ± is followed by an estimated margin of error under 95% confidence.

ducted a preliminary fine-tuning of the BART-large model on Oracle EDUs to assess the potential of the Extraction-Abstraction approach. This experiment presupposes an extraction module operating at peak efficiency, achieving a 100% F1 score, whereby all important EDUs are concatenated and the BART-large model is fine-tuned on this aggregated content. Intriguingly, as will be presented in Table 6.7, our findings reveal that this method does not yield higher ROUGE scores than simply applying BART to the original documents, presenting a significant insight into the limitations of an extraction summarization approach. However, we ran experiments of Table 6.4 to assess the effectiveness of different aspects of

Table 6.5: Comparing Our Best Model with Previous Models on CNN/DM.

Model	**R-1**	**R-2**	**R-L**
Oracle			
Lead3	40.42	17.62	36.67
Oracle (EDUs)	61.61	37.82	59.27
Extractive			
DiscoBERT (Xu et al., 2020)	43.38	20.44	40.21
DiscoBERT w Coref	43.58	20.64	40.42
DiscoBERT w RST	43.68	20.71	40.54
DiscoBERT w RST & Coref	**43.77**	20.85	40.67
BERTSUMEXT (Liu and Lapata, 2019)	43.25	20.24	39.63
Abstractive			
BERTSUMABS (Liu and Lapata, 2019)	41.72	19.39	38.76
BART-base† (Lewis et al., 2020)	41.44	18.49	38.32
Hybrid			
BERTSUMEXTABS (Liu and Lapata, 2019)	42.13	19.60	39.18
Longformer with MLP for RST and Coref (ours)	43.73	**20.91**	**40.72**

† We generated the result based on their provided model and hyperparameters on our CNN/DM testset.

our approach, including graph effectiveness or graph incorporation techniques.

6.2 Discussion of the results

In this section, we delve deeper into our findings, examining them across multiple key aspects. Our main goal for this study was to improve the baseline results while thoroughly testing our models. This led us to select a secondary challenging dataset with multiple contrasting attributes compared to CNN/DM. The XSum dataset, which is highly abstractive and concise, allowed us to more accurately evaluate our approach in comparison to other studies that might choose similar secondary datasets to reconfirm their initial results.

Regarding the use of graphs, we found that leveraging RST and Coref graphs improved the baseline version (which does not use any graph information) and, in the case of CNN/DM, our approach outperformed previous models. The method of incorporating graph information is critical. We operated our graph-based encoders under nearly identical set-

Table 6.6: Comparing Our Best Model with Previous Models on XSum.

Model	**R-1**	**R-2**	**R-L**
Oracle			
LEAD1 †	16.30	1.60	11.95
Oracle (Best Sentence) (Liu and Lapata, 2019)	29.79	8.81	22.66
Oracle (Best EDUs) (ours)	36.03	11.47	30.86
Abstractive			
BERTSUMABS (Liu and Lapata, 2019)	38.76	16.33	31.15
BART-large ‡ (Lewis et al., 2020)	**42.34**	**18.45**	**32.40**
Zeroshot BART-large on oracle EDUs	31.94	11.10	24.62
Finetuned BART-large on oracle EDUs	42.34	18.44	32.40
Hybrid			
BERTSUMEXTABS (Liu and Lapata, 2019)	38.81	16.50	31.27
Longformer with MLP for RST (ours)	36.82	14.66	28.67

† The term "Lead1" refers to the first sentences of the input documents.
‡ We generated this result based on their provided fine-tuned model.

tings. Despite various attempts, meticulous hyperparameter tuning, and significant resource allocation to GAT-based experiments, none achieved higher ROUGE scores in their final summaries. Testing GNNs with edge labels also did not improve performance. This suggests that the prevailing approach of using GNNs for incorporating graph information might not always be suitable, and a simple MLP could be more effective.

Encoder models reported in Table 6.1 and Table 6.2 are not the only ones we experimented with. For each model, we tested multiple combinations of hyperparameters and design choices. For example, the initial predecessor of the "Longformer without GAT" model achieved only a 35% F1 score until we optimized the learning rate scheduler, dropout layers, loss function, and optimizer weight. A notable technique involved assigning greater weight to positive labels. Given that approximately 10% of EDUs are labeled important in both the CNN/DM and XSum datasets, this approach addressed issues of slow convergence and overfitting in our imbalanced datasets.

We approached the fine-tuning of encoder models differently. For models using GAT, we fine-tuned them in two phases. First, we froze the weights of the language model and

trained the GAT layers. Then, we unfroze the language model weights and fine-tuned the entire network for a few epochs. However, this approach did not improve GAT performance. While most studies use only ROUGE scores to evaluate performance, we employed several additional evaluation metrics (see Appendix A.2) to facilitate comparison of our best model with future models. Regarding the language models in our encoders, we found that Longformer-based models generally performed better than their BERT-based counterparts.

Regarding our extraction-abstraction (hybrid) approach, our analysis indicates that an extraction approach based on greedy labeling might not be appropriate. Upon applying BART model in zero-shot setting on oracle EDUs of CNN/DM, we reached a higher ROUGE score than BART itself. However, this improvement was not observed when using the fine-tuned BART-large model on oracle EDUs from the XSum dataset, as detailed in Table 6.6 and numerically analyzed in Table 6.7. This discrepancy highlights the potential limitations of the extraction approach based on greedy labeling and underscores the need for further investigation into the effectiveness of different strategies across various models and datasets. It also raises questions about the adaptability and generalizability of these approaches, suggesting that what works well for one dataset or model might not necessarily translate to success with another.

6.3 Chapter Summary

In this chapter, we conducted a thorough evaluation of our proposed approach, examining the performance of different encoder models on the CNN/DM and XSum datasets, highlighting the impact of incorporating graph information and different architectures. Our findings revealed that while Longformer-based models generally achieved lower F1 scores compared to their BERT-based counterparts, they benefited from their ability to operate on long-range input documents. In these cases of having long input, graph information can also act as an extra guidance for the model to select more important parts. Comparing to GAT, MLP performed better in utilizing the graph information. We compared our

Table 6.7: Comparing the Performance of Extractive Approaches on CNN/DM and XSum.

Model	**R-1**	**R-2**	**R-L**
CNN/DM			
BART-base	41.44	18.49	38.32
Oracle (EDUs)	**61.61** (↑)	**37.82** (↑)	**59.27** (↑)
Zeroshot BART-base on oracle EDUs	53.43 (↑)	32.20 (↑)	51.05 (↑)
Finetuned BART-base on oracle EDUs	60.89 (↑)	37.59 (↑)	57.47 (↑)
XSum			
BART-large	**42.34**	**18.45**	**32.40**
Oracle (EDUs) (ours)	36.03 (↓)	11.47 (↓)	30.86 (↓)
Zeroshot BART-large on oracle EDUs	31.94 (↓)	11.10 (↓)	24.62 (↓)
Finetuned BART-large on oracle EDUs	42.34 (↓)	18.44 (↓)	32.40 (↓)

We set one abstractive model for each dataset as the base-line of comparison and compare ROUGE scores of oracle EDUs and Hybrid model on top of Oracle EDUs to see how far we can go in extraction approach direction. Since XSum is shorter than CNN/DM we were able to use the Large version of BART model for it.

best models with previous papers our baseline of comparison, demonstrating improvements on CNN/DM dataset. Our discussion provided insights into the challenges and potential limitations of various strategies, emphasizing the importance of careful model design and evaluation.

Chapter 7

Conclusion and Future Directions

This chapter concludes our research and summarize its findings. We will also delve into the future directions that could enhance our model's architecture and data processing techniques.

7.1 Conclusion

We have introduced a hybrid summarization model which uses BERT as well as Longformer. We, initially, intended to use GAT architecture to incorporate graph information for improving the performance. Despite hyperparameter search or testing multiple architectural changes, GAT did not improved the performance in our settings. Later, we used a MLP for incorporating graph information which increased the performance. On CNN/DM dataset, the MLP version of our model surpasses the performance of our baseline models (Lewis et al., 2020; Xu et al., 2020; Li et al., 2020).

We undertook a thorough examination of our model on different aspects. These aspects include different datasets, graph types, graph incorporation techniques, hyperparameter design, training methods, evaluation techniques, base language models, and summarization approaches. This multifaceted analysis aimed to understand better under what circumstances our approach might enhance performance.

In our work, we provided an annotated version of the XSum dataset enhanced with RST graphs and higher ROUGE scores for oracle EDUs. Our detailed data processing instructions are designed to facilitate further research, offering a roadmap for other researchers to

apply these methodologies to other datasets.

Our work lays a foundation for subsequent innovations in the field of document summarization, highlighting areas where further investigation and development could yield significant advancements.

7.2 Future Directions

Looking forward, we identify promising directions for future research, both in terms of data processing techniques and model architecture enhancements. These avenues could include exploring more sophisticated methods for data preprocessing, investigating alternative dataset employment, oracle labeling enhancements, and developing more advanced mechanisms for handling long documents in an end-to-end fashion.

There is one future work in improving the model's architecture and three directions regarding dataset preprocessing. We can use a novel idea that was used in Dialogue Management Systems to solve the gradient disconnection issue[1] in hybrid summarization models. Hosseini-Asl et al. (2022) in their work, SimpleTOD, showed that we can cast multiple modules needed in dialogue systems, such as intent detection, database query, or response generation, into one single model. This model, which is trained in an end-to-end manner[2], outperforms the previous stage-wise models as well as simplifies the training task as it does not involve multiple stages of training.

Figure 7.1, taken from their GitHub repository[3] under Creative Commons Licence, shows how they use one single language model, which is based on GPT, to handle all the phases of generating a final response in chatbot systems. Similarly, we can pass the input document, RST-graph, or other graph information to one language model to generate the index of the important EDUs, and then generate the final summaries, which are going to be

[1]The gradient disconnection issue, which interrupts gradient flow during back propagation, impedes learning. It often arises in architectures with separate modules or non-differentiable operations. In our case, each module is optimized individually, but their combination may not yield the best overall performance.

[2]End-to-end training refers to a training approach where a model is trained all at once for a specific task, without the presence of separately trained modules within the system.

[3]https://github.com/salesforce/simpletod

the important EDUs that the model has identified.

More specifically, we can first prepend the index of the EDUs to them, then wrap the whole list with <EDUs> and </EDUs> special tokens. After that, we can have any other information such as RST information which is similarly wrapped with two special tokens, such as <RstInfo> and </RstInfo>. The model, then, takes this input and generates the id of the important EDUs and the final summary. The output format would be:
<SelectedIds> $id_i\, id_j\, id_k$ </SelectedIds><summary> generated summary </summary>
in which the model has selected IDs i, j, and k as important IDs that we need to attend more in generating the rest of the sequence that is our summary.

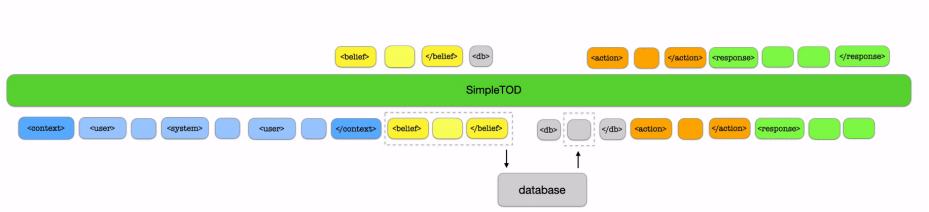


Figure 7.1: SimpleTOD, A Simple Language Model for Task-Oriented Dialogue, from (Hosseini-Asl et al., 2022) GitHub repository under Creative Commons licence.

The first and simplest idea for improving the processed XSum dataset, or any other future datasets, is to choose a semantic-aware metric, such as BERTscore, instead of a syntax-sensitive metric that is commonly used, like ROUGE. This is most likely to solve the issue of having an empty oracle list in Oracle EDU generation, presented in Table 4.6. Secondly, one can generate a Coref graph for XSum dataset as it might help in having a better representation of the documents.

The last and most costly idea is to generate the graphs for a data set that is lengthy. This option was unfeasible for us, due to its high computational needs that were not even achievable by using our Compute Canada allocated resources. This last direction, however, is the most promising. Current text summarization datasets, like CNN/DM, that were previously considered lengthy, by exceeding the maximum input size of many previous models such as BERT, now are easily handled by large models that can capture the big picture of documents

in an abstractive way. As we mentioned in Section 3.2, BART outperforms all the extractive models by incorporating two large models, more data, more pre-training tasks, and having double the size of BERT's maximum input length. These advancements shift our definition of "long document" and suggest applying the extractive and hybrid approaches for longer documents. In this case, other representation tools, like our graph structures, can aid the selection of information thereby making a meaningful improvement.

Bibliography

D. Anderson and G. McNeill. Artificial neural networks technology. *Kaman Sciences Corporation*, 258(6):1–83, 1992.

I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document transformer, 2020. URL https://arxiv.org/abs/2004.05150.

L. Carlson, D. Marcu, and M. E. Okurovsky. Building a discourse-tagged corpus in the framework of rhetorical structure theory. In *Proceedings of the SIGDIAL 2001 Workshop, The 2nd Annual Meeting of the Special Interest Group on Discourse and Dialogue*, Aalborg, Denmark, 2001.

D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2016.

M. P. Deisenroth, A. Faisal, and C. S. Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020. ISBN 9781108455145. URL https://www.cambridge.org/au/universitypress/subjects/computer-science/pattern-recognition-and-machine-learning/mathematics-machine-learning?format=PB.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, and J. Larson. From local to global: A graph rag approach to query-focused summarization, 2024.

W. S. El-Kassas, C. R. Salama, A. A. Rafea, and H. K. Mohamed. Automatic text summarization: A comprehensive survey. *Expert Systems with Applications*, 165: 113679, 2021. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2020.113679. URL https://www.sciencedirect.com/science/article/pii/S0957417420305030.

M. Eric, R. Goel, S. Paul, A. Sethi, S. Agarwal, S. Gao, A. Kumar, A. Goyal, P. Ku, and D. Hakkani-Tur. MultiWOZ 2.1: A consolidated multi-domain dialogue dataset with state corrections and state tracking baselines. In N. Calzolari, F. Béchet, P. Blache, K. Choukri, C. Cieri, T. Declerck, S. Goggi, H. Isahara, B. Maegaard, J. Mariani,

H. Mazo, A. Moreno, J. Odijk, and S. Piperidis, editors, *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 422–428, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4. URL https://aclanthology.org/2020.lrec-1.53.

D. Graff et al. English gigaword third edition. Web Download, 2007. LDC2007T07.

V. Gupta, P. Bharti, P. Nokhiz, and H. Karnick. SumPubMed: Summarization dataset of PubMed scientific articles. In J. Kabbara, H. Lin, A. Paullada, and J. Vamvas, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: Student Research Workshop*, pages 292–303, Online, Aug. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-srw.30. URL https://aclanthology.org/2021.acl-srw.30.

W. L. g. Hamilton. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159, 2020.

D. Harman and P. Over. The duc summarization evaluations, 2002-03-01 2002.

D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus), 2023.

E. Hosseini-Asl, B. McCann, C.-S. Wu, S. Yavuz, and R. Socher. A simple language model for task-oriented dialogue, 2022.

L. Hou, P. Hu, and C. Bei. Abstractive document summarization via neural model with joint attention. In X. Huang, J. Jiang, D. Zhao, Y. Feng, and Y. Hong, editors, *Natural Language Processing and Chinese Computing*, pages 329–338, Cham, 2018. Springer International Publishing. ISBN 978-3-319-73618-1.

Y. Ji and J. Eisenstein. Representation learning for text-level discourse parsing. In K. Toutanova and H. Wu, editors, *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13–24, Baltimore, Maryland, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/P14-1002. URL https://aclanthology.org/P14-1002.

D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR, USA, 1st edition, 2000. ISBN 0130950696.

F. Kalota. A primer on generative artificial intelligence. *Education Sciences*, 14(2), 2024. ISSN 2227-7102. doi: 10.3390/educsci14020172. URL https://www.mdpi.com/2227-7102/14/2/172.

T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks, 2017.

P. Koehn. Statistical significance tests for machine translation evaluation. In D. Lin and D. Wu, editors, *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 388–395, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://aclanthology.org/W04-3250.

J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1. URL http://dl.acm.org/citation.cfm?id=645530.655813.

V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.703. URL https://aclanthology.org/2020.acl-main.703.

Z. Li, W. Wu, and S. Li. Composing elementary discourse units in abstractive summarization. In D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6191–6196, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.551. URL https://aclanthology.org/2020.acl-main.551.

L. Liao, T. Zhu, L. H. Long, and T. S. Chua. Multi-domain dialogue state tracking with recursive inference. In *Proceedings of the Web Conference 2021*, WWW '21, page 2568–2577, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383127. doi: 10.1145/3442381.3450134. URL https://doi.org/10.1145/3442381.3450134.

C.-Y. Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://aclanthology.org/W04-1013.

B. Liu and L. Wu. Graph neural networks in natural language processing. In L. Wu, P. Cui, J. Pei, and L. Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 463–481. Springer Singapore, Singapore, 2022.

Y. Liu and M. Lapata. Text summarization with pretrained encoders. In K. Inui, J. Jiang, V. Ng, and X. Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3730–3740, Hong Kong, China, Nov. 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1387. URL https://aclanthology.org/D19-1387.

Y. Liu, P. Liu, D. Radev, and G. Neubig. Brio: Bringing order to abstractive summarization, 2022. URL https://arxiv.org/abs/2203.16804.

W. C. Mann and S. A. Thompson. Rhetorical structure theory: Toward a functional theory of text organization. *Text - Interdisciplinary Journal for the Study of Discourse*, 8(3): 243–281, 1988. doi: doi:10.1515/text.1.1988.8.3.243. URL https://doi.org/10.1515/text.1.1988.8.3.243.

C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky. The Stanford CoreNLP natural language processing toolkit. In K. Bontcheva and J. Zhu, editors, *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, Baltimore, Maryland, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/P14-5010. URL https://aclanthology.org/P14-5010.

E. Miltsakaki, R. Prasad, A. Joshi, and B. Webber. The penn discourse treebank. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04)*, 2004.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 807–814, 2010.

R. Nallapati, B. Zhou, C. dos Santos, Ç. Gulçehre, and B. Xiang. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In S. Riezler and Y. Goldberg, editors, *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany, Aug. 2016. Association for Computational Linguistics. doi: 10.18653/v1/K16-1028. URL https://aclanthology.org/K16-1028.

S. Narayan. The generalized sigmoid activation function: Competitive supervised learning. *Information Sciences*, 99(1):69–82, 1997. ISSN 0020-0255. doi: https://doi.org/10.1016/S0020-0255(96)00200-9. URL https://www.sciencedirect.com/science/article/pii/S0020025596002009.

S. Narayan, S. B. Cohen, and M. Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, 2018.

L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. In *Proceedings of the 7th International World Wide Web Conference (WWW)*. Stanford InfoLab, 1999.

J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In M. Walker, H. Ji, and A. Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1202. URL https://aclanthology.org/N18-1202.

M. Popel, M. Tomkova, J. Tomek, Ł. Kaiser, J. Uszkoreit, O. Bojar, and Z. Žabokrtský. Transforming machine translation: a deep learning system reaches news translation quality comparable to human professionals. *Nature Communications*, 11(1):4381, Sep 2020. ISSN 2041-1723. doi: 10.1038/s41467-020-18073-9. URL https://doi.org/10.1038/s41467-020-18073-9.

L. Pourkarimi, M. Yaghoobi, and M. Mashinchi. Efficient curve fitting: An application of multiobjective programming. *Applied Mathematical Modelling*, 35(1):346–365, 2011. ISSN 0307-904X. doi: https://doi.org/10.1016/j.apm.2010.06.009. URL https://www.sciencedirect.com/science/article/pii/S0307904X10002374.

A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. 2018. URL https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf.

M. K. Rohil and V. Magotra. An exploratory study of automatic text summarization in biomedical and healthcare domain. *Healthcare Analytics*, 2:100058, 2022. ISSN 2772-4425. doi: https://doi.org/10.1016/j.health.2022.100058. URL https://www.sciencedirect.com/science/article/pii/S2772442522000223.

Q. Ruan, M. Ostendorff, and G. Rehm. Histruct+: Improving extractive text summarization with hierarchical structure information, 2022.

S. Ruder. An overview of gradient descent optimization algorithms, 2017.

E. Sandhaus. The new york times annotated corpus. Web Download, 2008. LDC2008T19.

M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45:2673–2681, November 1997.

A. See, P. J. Liu, and C. D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, 2017.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

R. Sukthanker, S. Poria, E. Cambria, and R. Thirunavukarasu. Anaphora and coreference resolution: A review, 2018.

T. van Laarhoven. L2 regularization versus batch and weight normalization, 2017.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017. URL https://arxiv.org/abs/1706.03762.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks, 2018.

S. Wang, X. Zhao, B. Li, B. Ge, and D. Tang. Integrating extractive and abstractive models for long text summarization. In *2017 IEEE International Congress on Big Data (BigData Congress)*, pages 305–312, 2017. doi: 10.1109/BigDataCongress.2017.46.

Y. Wang, S. Li, and J. Yang. Toward fast and accurate neural discourse segmentation. In E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 962–967, Brussels, Belgium, Oct.-Nov. 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1116. URL https://aclanthology.org/D18-1116.

F. Wolf and E. Gibson. Representing discourse coherence: A corpus-based study. *Computational Linguistics*, 31(2):249–287, 2005.

L. Wu, P. Cui, J. Pei, L. Zhao, and L. Song. Graph neural networks. In L. Wu, P. Cui, J. Pei, and L. Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 27–37. Springer Singapore, Singapore, 2022.

J. Xu, Z. Gan, Y. Cheng, and J. Liu. Discourse-aware neural extractive text summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020.

J. Zhang, Y. Zhao, M. Saleh, and P. Liu. PEGASUS: Pre-training with extracted gap-sentences for abstractive summarization. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 11328–11339. PMLR, 13–18 Jul 2020a. URL https://proceedings.mlr.press/v119/zhang20ae.html.

T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi. Bertscore: Evaluating text generation with bert, 2020b.

C. Zhu, W. Hinthorn, R. Xu, Q. Zeng, M. Zeng, X. Huang, and M. Jiang. Enhancing factual consistency of abstractive summarization. In K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tur, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, and Y. Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 718–733, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.58. URL https://aclanthology.org/2021.naacl-main.58.

Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.

Appendix A

A.1 Sample files of the dataset generation pipeline

In this part, we use document indexed '1' from our processed testset, Figure A.1, to showcase different type of files that we have in the pipeline. This file is the input of both CoreNLP, to generate .xml file (Figure A.3), and NerualEDUseg, to generate the EDUs (Figure A.2). After that, we generated .XML files, and we generated a simplified version of that in .conll file (Figure A.4). This file contains the attributes of tokens, discussed earlier in Section 4.2.1.

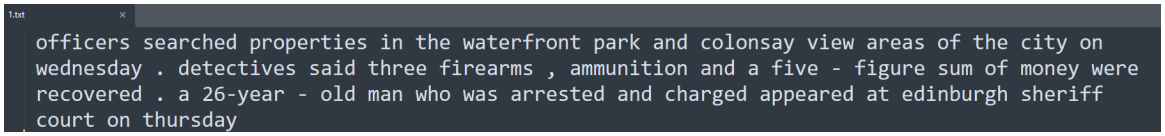


Figure A.1: Sample raw input text (1.txt)

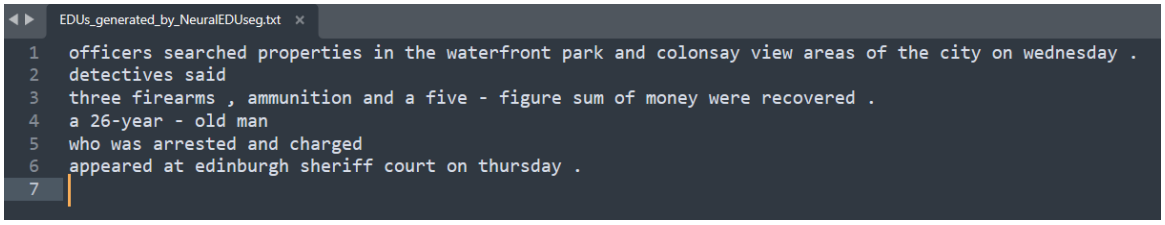


Figure A.2: NeuralEDUseg output on 1.txt raw input.

As mentioned before, DPLP is not highly accurate in EDU segmentation; hence, we use NeuralEDUseg for segmentation and override the EDU ID of tokens that are generated by DPLP in .merge files. By comparing Figure A.5 and Figure A.6 we can see this difference.

The next step is generating the .bracket file, shown in Figure A.7, from the modified .merge file. This .bracket file, by DPLP notes in their codes, should contain RST tree. However, the specific format of RST tree that they are generating is not the same format as illustrated in a recent paper, (Xu et al., 2020). As we mentioned before, the notation and terminology of RST tree have gone through many changes and DPLP's format is not the same as the original paper, in 1980s. We leave the examination of changes in the RST tree terminology for interested readers.

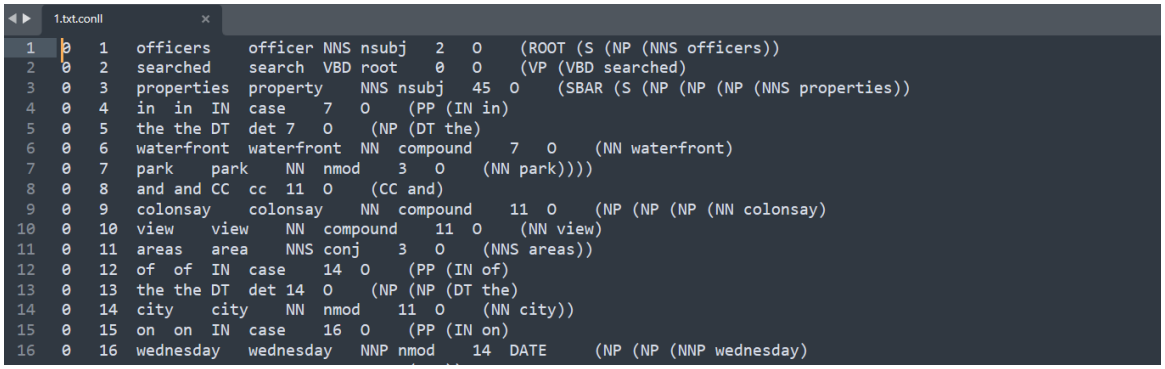
Finally, from modified .merge file and .bracket files, we generate RST graph, illustrated in Figure A.8. This is a linearized format of RST graph, and in this case, it seems that DPLP is doing a simplification and only using a subset of RST tree labels, as the labels

1.txt.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="CoreNLP-to-HTML.xsl" type="text/xsl"?>
<root>
  <document>
    <docId>1.txt</docId>
    <sentences>
      <sentence id="1" line="1">
        <tokens>
          <token id="1">
            <word>officers</word>
            <lemma>officer</lemma>
            <CharacterOffsetBegin>1</CharacterOffsetBegin>
            <CharacterOffsetEnd>9</CharacterOffsetEnd>
            <POS>NNS</POS>
            <NER>O</NER>
          </token>
          <token id="2">
            <word>searched</word>
            <lemma>search</lemma>
            <CharacterOffsetBegin>10</CharacterOffsetBegin>
            <CharacterOffsetEnd>18</CharacterOffsetEnd>
            <POS>VBD</POS>
            <NER>O</NER>
          </token>
          <token id="3">
            <word>properties</word>
            <lemma>property</lemma>
            <CharacterOffsetBegin>19</CharacterOffsetBegin>
            <CharacterOffsetEnd>29</CharacterOffsetEnd>
            <POS>NNS</POS>
            <NER>O</NER>
          </token>
          <token id="4">
            <word>in</word>
            <lemma>in</lemma>
            <CharacterOffsetBegin>30</CharacterOffsetBegin>
            <CharacterOffsetEnd>32</CharacterOffsetEnd>
            <POS>IN</POS>
            <NER>O</NER>
          </token>
```

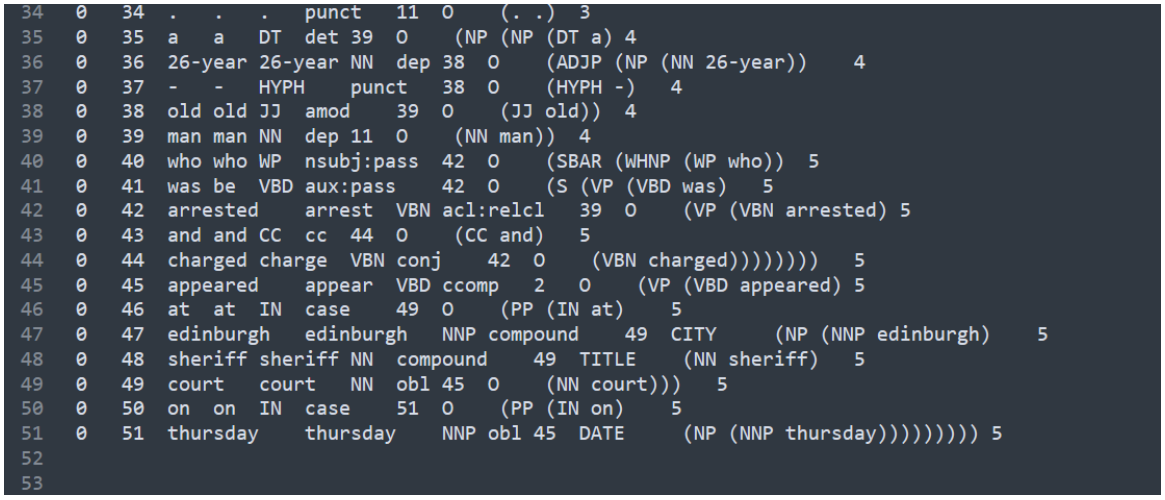
Figure A.3: Sample .XML file (Generated from 1.txt).

used in the graph are not exactly the same as the tree. Now, by examining the labels and drawing connections of RST graph, we can see that it represents the original raw document in a more informative way and all the labels of the connections are logical.



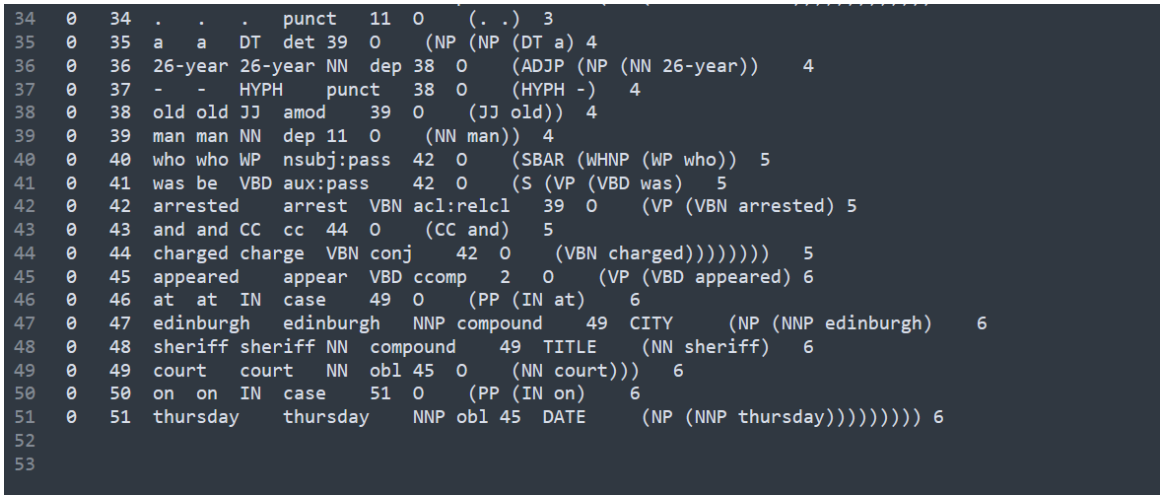
```
1.txt.conll
0  1  officers    officer NNS nsubj   2  O    (ROOT (S (NP (NNS officers))
0  2  searched    search  VBD root    0  O    (VP (VBD searched)
0  3  properties  property    NNS nsubj   45  O    (SBAR (S (NP (NP (NP (NNS properties))
0  4  in  in  IN  case    7   O    (PP (IN in)
0  5  the the DT  det 7   O    (NP (DT the)
0  6  waterfront  waterfront  NN  compound    7   O    (NN waterfront)
0  7  park    park    NN  nmod    3   O    (NN park))))
0  8  and and CC  cc  11  O    (CC and)
0  9  colonsay    colonsay    NN  compound    11  O    (NP (NP (NP (NN colonsay)
0  10 view    view    NN  compound    11  O    (NN view)
0  11 areas   area    NNS conj    3   O    (NNS areas))
0  12 of  of  IN  case    14  O    (PP (IN of)
0  13 the the DT  det 14  O    (NP (NP (DT the)
0  14 city    city    NN  nmod    11  O    (NN city))
0  15 on  on  IN  case    16  O    (PP (IN on)
0  16 wednesday   wednesday   NNP nmod    14  DATE     (NP (NP (NNP wednesday)
```

Figure A.4: Sample .conll file (Generated from 1.XML).



```
0  34 .   .   .   punct   11  O    (. .)  3
0  35 a   a   DT  det 39  O    (NP (NP (DT a) 4
0  36 26-year 26-year NN  dep 38  O    (ADJP (NP (NN 26-year))    4
0  37 -   -   HYPH    punct   38  O    (HYPH -)   4
0  38 old old JJ  amod    39  O    (JJ old))  4
0  39 man man NN  dep 11  O    (NN man))  4
0  40 who who WP  nsubj:pass  42  O    (SBAR (WHNP (WP who))  5
0  41 was be  VBD aux:pass    42  O    (S (VP (VBD was)   5
0  42 arrested    arrest  VBN acl:relcl   39  O    (VP (VBN arrested) 5
0  43 and and CC  cc  44  O    (CC and)   5
0  44 charged charge  VBN conj    42  O    (VBN charged))))))))   5
0  45 appeared    appear  VBD ccomp   2   O    (VP (VBD appeared) 5
0  46 at  at  IN  case    49  O    (PP (IN at)    5
0  47 edinburgh   edinburgh   NNP compound    49  CITY     (NP (NNP edinburgh)    5
0  48 sheriff sheriff NN  compound    49  TITLE    (NN sheriff)   5
0  49 court   court   NN  obl 45  O    (NN court)))   5
0  50 on  on  IN  case    51  O    (PP (IN on)    5
0  51 thursday    thursday    NNP obl 45  DATE     (NP (NNP thursday)))))))))) 5
```

Figure A.5: Sample .merge file (Generated from 1.XML by DPLP).



```
0  34 .   .   .   punct   11  O    (. .)  3
0  35 a   a   DT  det 39  O    (NP (NP (DT a) 4
0  36 26-year 26-year NN  dep 38  O    (ADJP (NP (NN 26-year))    4
0  37 -   -   HYPH    punct   38  O    (HYPH -)   4
0  38 old old JJ  amod    39  O    (JJ old))  4
0  39 man man NN  dep 11  O    (NN man))  4
0  40 who who WP  nsubj:pass  42  O    (SBAR (WHNP (WP who))  5
0  41 was be  VBD aux:pass    42  O    (S (VP (VBD was)   5
0  42 arrested    arrest  VBN acl:relcl   39  O    (VP (VBN arrested) 5
0  43 and and CC  cc  44  O    (CC and)   5
0  44 charged charge  VBN conj    42  O    (VBN charged))))))))   5
0  45 appeared    appear  VBD ccomp   2   O    (VP (VBD appeared) 6
0  46 at  at  IN  case    49  O    (PP (IN at)    6
0  47 edinburgh   edinburgh   NNP compound    49  CITY     (NP (NNP edinburgh)    6
0  48 sheriff sheriff NN  compound    49  TITLE    (NN sheriff)   6
0  49 court   court   NN  obl 45  O    (NN court)))   6
0  50 on  on  IN  case    51  O    (PP (IN on)    6
0  51 thursday    thursday    NNP obl 45  DATE     (NP (NNP thursday)))))))))) 6
```

Figure A.6: Sample modified .merge file overridden by NeuralEDUseg.

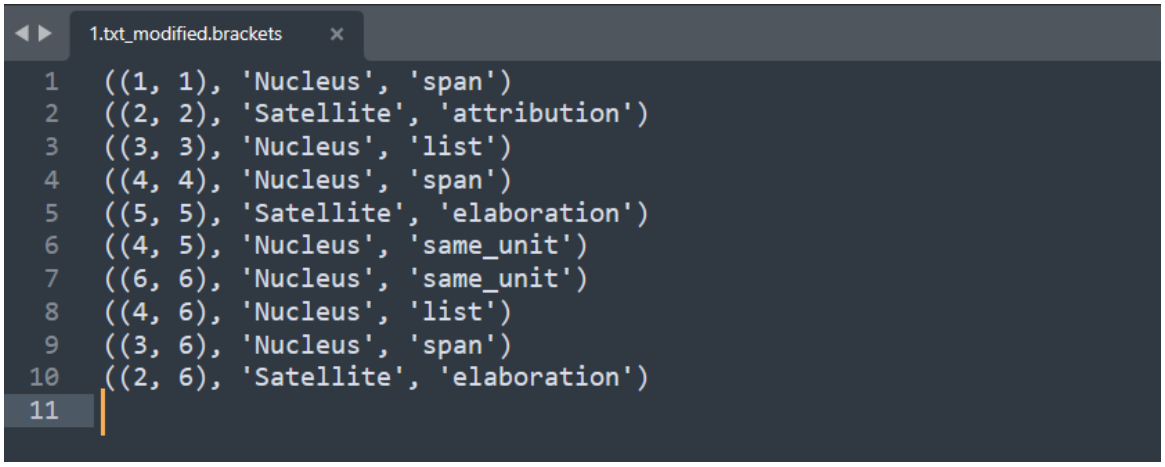


Figure A.7: Sample .bracket file.

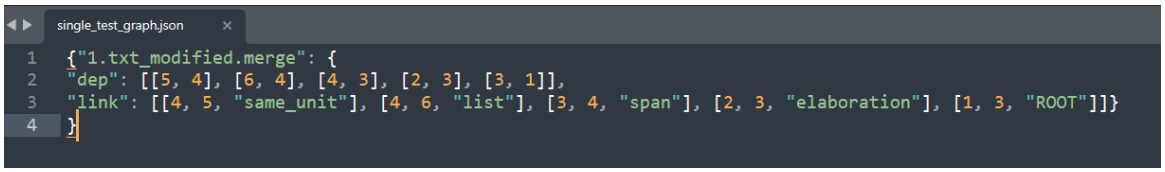


Figure A.8: Sample final RST graph json file.

A.2 Additional Evaluation

In this section, we delve into additional evaluation metrics to assess the quality of the generated summaries within the test dataset. Table A.1 showcases the proportions of novel N-grams across different text types relative to their original source texts. This analysis reveals that human-generated summaries (refereed to as reference summaries) exhibit a greater ability to coin new terms and apply creative expression to encapsulate concepts uniquely in their own words. As our decoder module is able to generate new words other than the word in the input, following the reference summaries, predicted summaries display high novel N-gram proportions. Conversely, Oracle EDUs and Selected EDUs, on account of their extractive nature, show no novelty in 1-grams but manifest some innovation in bigrams and trigrams. This introduction of novel bigrams and trigrams, despite their extractive basis, is attributed to the novel concatenations of segments that were not sequentially adjacent in the original text.

Table A.1: Novel N-grams Proportions

Metric	n=1	n=2	n=3
Oracle EDUs	0.000	0.015	0.071
Selected EDUs	0.000	0.012	0.053
Predicted Summaries	0.005	0.060	0.171
Reference Summaries	0.008	0.173	0.491

We continue our evaluation with two other automatic metrics; BERT-score[1] and BART-score[2]. Our base-line models did not report these metrics on thier best model, yet we calculated them and reported them so that other future works can compare their performance on these metrics with us. Initially, we also planned to use BLUE score. This score is for machine-translation and not relevant to our task, however.

Table A.2: Sample Metrics Table2

Metric	BERT Score	BART Score
Our Best Model	0.87	-3.73

So far, we have relied on an automatic evaluation approach. At this stage, we will examine a single sample summary, presented in Table A.3, from the perspectives of coherence and cohesion. Both coherence and cohesion are crucial in writing, ensuring that the text is understandable and flows smoothly. While cohesion deals with the links between words and sentences, coherence focuses on the logical flow and organization of ideas within a text. As demonstrated in the table, our generated summary closely resembles human summarization by correctly weaving ideas together. However, selective approaches tend to fall short on these metrics, even if they may achieve high scores with automatic evaluation methods.

[1]https://pypi.org/project/bert-score/.

[2]https://github.com/neulab/BARTScore/tree/main

Table A.3: Sample Summary

Type	**Value**
Source Document	(cnn) the dark knight returns ... again . “ the dark knight returns , ” published in 1986 , is widely credited for resurrecting batman in pop culture , something we ’ve seen referenced in everything from 1989 ’s “ batman ” to the “ dark knight ” trilogy and the upcoming “ batman v. superman : dawn of justice . ” now popular comic book writer frank miller is returning to his best-known story . dc comics (a time warner company , like cnn) announced friday the final chapter in his “ the dark knight returns ” trilogy , in the form of “ the dark knight iii : the master race ” (“ the dark knight strikes again ” was released in 2001) . this third chapter in the grim saga will be released sometime in the fall . “ batman remains my favorite comic book hero and a sequel to dark knight is going to be daunting , ” said miller in a press release , “ but we ’ll do our best . ” miller will be joined by acclaimed artist brian azzarello ..
Selected EDUs	the dark knight returns ... again . “ the dark knight returns , ” published in 1986 , is widely credited for resurrecting batman in pop culture , now popular comic book writer frank miller is returning to his best-known story . announced friday the final chapter in his “ the dark knight returns ” trilogy , in the form of “ the dark knight iii : the master race ” this third chapter in the grim saga will be released sometime in the fall . miller will be joined by acclaimed artist brian azzarello .
Oracle EDUs	“ the dark knight returns , ” now popular comic book writer frank miller is returning to his best-known story . dc comics
Generated summary	“the dark knight returns” is widely credited for resurrecting batman in pop culture popular comic book writer frank miller is returning to his best-known story miller will be joined by acclaimed artist brian azzarello this third chapter in the grim saga will be released sometime in the fall
Reference summary	classic comic book “ the dark knight returns” is getting a second sequel legendary comics writer frank miller is returning to the story that made him famous.