4-10-2023

A Deep Learning Approach to Extractive Text Summarization Using Knowledge Graph and Language Model

Yichen Gao
Illinois State University, ycgao98@gmail.com

A DEEP LEARNING APPROACH TO EXTRACTIVE TEXT SUMMARIZATION USING KNOWLEDGE GRAPH AND LANGUAGE MODEL

YICHEN GAO

39 Pages

Extractive summarization has been widely studied, but the summaries generated by most current extractive summarization works usually disregard the article structure of the source document. Furthermore, the produced summaries are sometimes not representative sentences in the article. In this thesis, we propose an extractive summarization algorithm with knowledge graph enhancement that leverages both the source document and a knowledge graph to predict the most representative sentences for the summary. The aid of knowledge graphs enables deep learning models with pre-trained language models to focus on article structure information in the process of generating extractive summaries. Our proposed method has an encoder and a classifier: the encoder encodes the source document and the knowledge graph separately. The classifier inter-encodes the encoded source document and knowledge graph information by the cross-attention mechanism. Then the classifier determines whether the sentences belong to summary sentences or not. The results show that our model produces higher ROUGE scores on the CNN/Daily Mail dataset than the model without the knowledge graph for assistance, compared to the extractive summarization work based on the pre-trained language model.

KEYWORDS: Extractive summarization; Knowledge graph; Deep learning models

A DEEP LEARNING APPROACH TO EXTRACTIVE TEXT SUMMARIZATION USING KNOWLEDGE GRAPH AND LANGUAGE MODEL

YICHEN GAO

A Thesis Submitted in Partial
Fulfillment of the Requirements
for the Degree of

MASTER OF SCIENCE

School of Information Technology

ILLINOIS STATE UNIVERSITY

2023

A DEEP LEARNING APPROACH TO EXTRACTIVE TEXT SUMMARIZATION USING KNOWLEDGE GRAPH AND LANGUAGE MODEL

YICHEN GAO

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Dr. Hyoil Han, for her unwavering support, guidance, and mentorship throughout my thesis journey. Her invaluable insights and encouragement have been instrumental in shaping my research and academic view. I am truly grateful for her tolerance, diligence, and the many informative conversations that deepened my insight into our research topic. I am very fortunate to have learned and grown under her direction, and I am eternally grateful for all she has taught me.

I would also like to extend my appreciation to my committee members for their time and feedback. Their knowledge and insightful inquiries have been invaluable in helping me hone my research and analysis techniques.

I am grateful to my parents for their love, support, and encouragement as I pursued my education. Their unwavering confidence in my abilities and constant guidance has enabled me to overcome the challenges I have met. They have also been a source of motivation and inspiration. Their understanding and patience have played an important role in my study and growth.

Finally, I am grateful to the School of Information Technology for their support of the course instruction, graduate research opportunities, and computing resources.

Y.G.

CONTENTS

TABLES

FIGURES

CHAPTER I: INTRODUCTION

Automatic text summarization is an essential task in natural language processing, which aims to generate a concise and meaningful representation of a longer source document or a collection of documents. There are two approaches to summarizing text: extractive summarization and abstractive summarization. Extractive summarization extracts existing sentences containing the most critical content from the original text and makes these sentences a summary highlighting the original text. The abstractive extraction works more like a human. The abstractive summarization may not be part of the source document, but it still retells the critical information. In our research, we focus on a single document extractive summarization task.

In the extractive summarization experiment, the CNN/DM dataset (Nallapati, Zhou, dos Santos, Gủlçehre, & Xiang, 2016) is widely used for evaluating the summarization quality. The sentence labels used for extractive summaries in CNN/DM are generated by a greedy algorithm that maximizes the ROUGE score between the source document and the sentences in the golden summary. The summaries generated by this approach do not take into consideration the document's structural information of the text, such as which sentences represent the main content of the entire document content. Sometimes, sentences with high ROUGE scores do not belong to the optimal summaries (Narayan, Cohen, & Lapata, 2018).

In our work, we apply a knowledge graph to provide document structure. We first extract the important entity and relation data from the document, such as subject, predicate, and object. Then we design an algorithm that filters the redundant parts of the extracted data from the document and aggregates the remaining data. Furthermore, the processed data is reassembled into a knowledge graph of the corresponding document.

We take both the source document and the knowledge graph for extractive summarization, and we encode and process the source document through a pre-training language model. The model can obtain the corresponding textual and contextual information by processing the source document with a large-scale pre-trained language model. After that, we use graph neural networks to analyze and infer the knowledge graphs. Then we use two cross-attention transformer neural networks to cross-encode the features extracted from the knowledge graph with the source document to bring the text structure information for the extractive summarization task. With this approach, we can obtain higher-quality summaries. Our model performs better than the current optimal model in the same experimental setting and setup.

CHAPTER II: RELATED WORK

Extractive Summarization

Text summarization has been studied for decades. One of the earliest approaches to extractive summarization was TextRank (Mihalcea & Tarau, 20004). TextRank represents the document as a graph, where nodes represent sentences and edges represent the similarity between sentences. The algorithm scores each sentence based on the graph structure and selects the top-scoring sentences as the summary. ClusterRank (Garg, Favre, Reidhammer, & Hakkani T ür, 2009) also proposes a graph-based algorithm. ClusterRank focus on meeting summarization, which uses cluster analysis to identify important topics and participants, then ranks them based on their relevance to the meeting, and then selects the highest-scoring sentences from the most critical clusters to form the summary. LexRank (Erkan & Radev, 2004) applies an unsupervised approach to identify and rank the most important sentences in a document based on their similarity among sentences. OTExtSum (Tang, et al., 2022) uses optimal transport for extractive text summarization. The paper proposes using the Wasserstein distance, a measure of dissimilarity between probability distributions, to identify the most informative sentences in a text and generate a summary that preserves the vital information.

Many deep learning models have recently been proposed for text summarization tasks. BERT (Devlin, Chang, Lee, & Toutanova, 2018) is a pre-trained neural network for natural language tasks. BERT, a transformer-based architecture, has achieved state-of-the-art results on various NLP benchmarks, which prove the ability of BERT to capture the semantic and contextual information of the input text. RoBERTa (Liu, et al., 2019) is a variation of BERT and improves the pre-training process with longer sequences of text, dynamic masking, and removal of the next sentence prediction objective, which are not included in BERT. Compared to BERT, RoBERTa was used for extracting text summarization tasks when it was proposed, proved its effectiveness in generating high-quality summaries, and achieved state-of-the-art results in several NLP benchmark tests. Since the excellent performance of BERT/Roberta on NLP tasks, there is a lot of extension work based on the BERT/RoBERTa model.

BertSum (Liu, 2019) utilizes BERT pre-trained language models to boost the performance of extractive summaries. The approach involves fine-tuning BERT on a large corpus of text data to generate summaries and test the performance with various classifiers for sentence summarization. They get a good performance with the sentence Transformer layer. Narayan et al. (Narayan, Cohen, & Lapata, 2018) use a pre-trained RoBERTa model to generate sentence embeddings and train a policy network using reinforcement learning to rank sentences in the input text based on their importance for the summary. The author also designs a new reward loss function that takes care of the content and diversity of the abstract. The MatchSum (Zhong, et al., 2020) model applies a text-matching framework to consider the relationship between the generated summary and the input document. MatchSum determines the most informative summary as the final summary based on the relevance of multiple candidate summaries to the input document. This method has shown promising results, indicating the potential of text-matching techniques for improving the quality of extractive summarization. When summarizing, extracting whole sentences can result in the summary including unneeded and redundant information. Zhou et al. (Zhou, Wei, & Zhou, 2020) propose an extractive summarization method that uses subsentences as the extractive unit. They investigate the impact of different levels of granularity - word, sentence, and paragraph for sentence selection in extractive document summarization and evaluate their effectiveness on several benchmark datasets using a BERT-based model. Their result suggests that summarizing at the sentence level is more effective for extractive summarization than summarizing at the word or paragraph levels. Yuan et al. (Yuan, Wang, & Li, 2020) propose a hierarchical graph mask approach for fact-level extractive summarization using BERT-based models. They proposed an algorithm that splits sentences into multiple facts, capturing the relationships between entities and facts in the text. They then use a neural network to generate a hierarchical mask that identifies the most essential facts to be included in the summary. This method achieves state-of-the-art performance on several benchmark datasets, demonstrating the effectiveness of fact-level summarization for this task.

Text Summarization with Graph Algorithms

Graph algorithms have been used in the field of text summarization. Tan et al. (Tan, Wan, & Xiao, 2017) propose a document graph representation in which nodes represent sentences and weighted edges represent the semantic similarity between sentences. They then use an attention mechanism to capture the relationships between sentences and generate abstract summaries. Vhatkar et al. (Vhatkar, Bhattacharyya, & Arya, 2020) extract triples from sentences and encode triples in a graph structure to improve the document's semantic representation of sentences, then applies a neural network to determine the triples' score. Summary sentences are selected based on the highest score triples in the neural network. Wu et al. (Wu, Koncel-Kedziorski, Ostendorf, & Hajishirzi, 2020) propose a method for extracting summary graphs from long documents to produce a structured and condensed representation of the primary content. This study demonstrates the potential of integrating extractive summarization techniques with knowledge graph construction to generate a structured and interpretable summary of lengthy documents. The methodology provides valuable insights for future research on extractive summarization and knowledge graph techniques. Jin et al. (Jin, Wang, & Wan, 2020) apply a multi-granularity interaction network (MGSum) for multi-document summarization that combines extractive and abstractive approaches. MGSum is a hierarchical model with an extractive and abstractive module that considers document, paragraph, and sentence levels of granularity. It captures the interactions between these granularities so that the relationships within and between multiple documents can be understood better to produce better summarizations. Belwal et al. (Belwal, Rai, & Gupta, 2021) present a new graph-based extractive text summarization method using keyword modeling or topic modeling. The research team develops an algorithm that recognizes keywords or subjects within the input text and then uses this information to generate a weighted similarity network. By recording semantic linkages among sentences and between the sentences and whole documents, the graph is evaluated to calculate significance scores for each sentence. Afterward, the most pertinent sentences are chosen for the summary. Gokhan et al. (Gokhan, Smith, & Lee, 2022) developed GUSUM, an unsupervised graph-based method for extractive text summarization. Combining sentence feature scoring and Sentence-BERT embeddings, GUSUM generates a similarity graph representing the input text. A clustering algorithm is then applied to the graph to group similar sentences, and representative sentences are chosen for the summary. Zhu et al. (Zhu, et al., 2020) address the problem of factual consistency in abstractive summarization. They offer a strategy that extracts triples from the document and then represents them as a knowledge graph by processing each triple as a node. Then the model merged knowledge graph information into the model's decoder to motivate the model to produce summaries that maintain factual consistency with the source document. Their work highlights the potential of leveraging knowledge graphs to improve the quality and factual accuracy of generated summaries. Huang et al. (Huang, Wu, & Wang, 2020) present a novel approach for abstractive summarization that incorporates knowledge graphs to enhance the semantic understanding of the input text. They propose Abstractive Summarization with Graph Augmentation and semantic-driven RewarD (ASGARD), which integrates external knowledge from knowledge graphs into the summary generation process. Huang et al. (Huang, Wu, & Wang, 2020) introduce a semantic-driven cloze reward that evaluates the model's performance based on the ability to answer cloze-style questions derived from the source text and the knowledge graph to encourage the generation of semantically coherent and factually accurate summaries. Wang et al. (Wang, Liu, Zheng, Qiu, & Huang, 2020) propose a heterogeneous graph-based neural network for extractive summarization (HeterSumGraph). The authors construct a heterogeneous graph representation of the input document, where nodes represent different types of textual elements such as sentences, paragraphs, and entities, and edges capture various relationships among these elements. The model captures local and global contextual information by incorporating different types of nodes and relationships, improving sentence selection for extractive summarization.

CHAPTER III: DATA PREPROCESSING

The CNN/Daily Mail (Hermann, et al., 2015) dataset is publicly available for the text summarization task (Nallapati, Zhou, dos Santos, Gὺlçehre, & Xiang, 2016). We use the CNN/DM dataset for experimentation and evaluation in our extractive summarization experiment. The dataset contains more than 300,000 English news releases written by professional journalists from CNN and Daily Mail. Every document in the dataset corresponds to a single data item which contains information in the form of data fields. The first field is "id," a string containing the hexadecimal form for the SHA1 hash of the URL from which the story was retrieved. The second field is "article," which is a string that contains the body of the news article. Finally, the third field is "highlights," which is a string containing the highlight of the article as written by the article author. Each file name in the CNN/DM dataset is a sixteen-bit SHA1 hash code corresponding to the news article's address. The content of each file contains the news article along with its associated highlights. Each highlight is preceded by the "@ highlight "symbol to distinguish the highlights from the news article. Notably, each file contains multiple highlights, indicating the presence of multiple key points within the news article. This feature makes the dataset particularly useful for text summarization tasks.

Table 1 is an instance data file from the CNN/DM dataset, which includes the article's unique id and content from its original file. Tables 2 and 3 provide information about the CNN/DM dataset. Table 2 shows the average number of tokens in the articles and highlights of the dataset. The average number of tokens for the articles is 781, while the average for the highlights is 56. In the task of text summarization by machine learning, the dataset needs to be partitioned into a training set, a validation set, and a test set. The CNN/DM dataset also contains the source URLs of all the news in the dataset. These URLs form a one-to-one correspondence with the ids of the data and are partitioned into a training set, a validation set, and a test set. The preprocessing follows the segmentation of the dataset, as shown in Table 3. The dataset contains 312,085 instances, of which 287,113 are in the training set, 13,368 in the validation set, and 11,490 in the test set.

Table 1

An instance data file from the CNN/DM dataset

File Name (id)	0000bf554ca24b0c72178403b54c0cca62d9faf8
Content	Nicholas Levene, 48, was jailed for 13 years last November after he admitted orchestrating a lucrative Ponzi scheme that raked in £316million.
News article	…… He would take from Peter to pay Paul and move the funds between accounts in the financial havens of Jersey, Switzerland and Israel.
highlight	……. @highlight Nicholas Levene must pay the nominal sum because he is bankrupt @highlight ……

Table 2

Average Token Count of Articles and Highlights

Data Field	Average Token Count
Article	781
Highlights	56

Table 3

Data Split of CNN/DM Dataset

Dataset number	Number of Instances
Dataset	312085
Train	287113
Validation	13368
Test	11490

Article and Highlight Split

Several preprocessing steps must be taken to prepare the raw CNN/Daily Mail dataset for a large pre-trained NLP neural network. One crucial step is tokenization, which involves splitting the text into individual words or subwords known as tokens. Tokenization helps the neural network understand the structure of the text. The Stanford CoreNLP toolkit (Manning, et al., 2014) is utilized to perform tokenization. This toolkit is an open-source natural language processing toolkit developed by the Stanford NLP Group. In addition to tokenization, the CoreNLP toolkit offers a variety of tools for processing natural language text, including lexical annotation, named entity recognition, sentiment analysis, and coreference parsing. It is commonly used in industry and academia for many natural language processing tasks, such as sentiment analysis, information extraction, and machine translation.

After using Stanford CoreNLP to tokenize the raw text from the CNN/Daily Mail dataset, we obtain a JSON file that contains the tokenized sentences. Each sentence is treated as a separate line of text and is stored as a list of individual words. To extract the source text and the corresponding summary from each file in the CNN/DM dataset (as shown in Table 1), we iterate through each sentence in the JSON file, adding each word of the sentence to the source text until it encounters a sentence that starts with "@highlight." If the first word of the sentence is "@highlight," the subsequent content is considered a summary and is added to the corresponding list of summaries. Finally, the source text and summaries are saved in a new JSON file.

Summarization Extraction

Two common types of summarization tasks are extractive and abstractive summarization. In extractive summarization, the goal is to select a subset of sentences from the source document that reflects the essential information. In abstractive summarization, the goal is to generate a summary that contains the main points of the source document concisely and coherently.

In the CNN/DM dataset, the source documents are the news articles themselves, and the highlights generated by the article authors are used as human-generated abstractive summaries. (Nallapati, Zhou, dos Santos, Gủlçehre, & Xiang, 2016). For extractive summaries, select up to three sentences from the source document that overlap the most with the words in the abstractive summary (Liu, 2019). These sentences are then used as extractive summaries reflecting the content of the source document.

The ROUGE metric (Lin & Och, 2004) is a commonly used metric for evaluating the quality of text summarization in the extractive summarization task. In this approach, the goal is to select a subset of sentences from the source document that best captures the main ideas of the text. To achieve this, Liu et al. use the greedy algorithm to iterate through the unselected sentences of the source document one at a time (Liu, 2019).

When building extractive summaries from the source text, the basic unit of the summary is the sentence. Let S be the set of all sentences in the news article and let S' be the subset of sentences that have been selected as summary sentences. The set of unselected sentences in the original text is the difference set, $S - S'$, which represents the sentences that have not yet been added to the summary.

To select the next sentence to be added to the summary, we calculate the sum of Rouge-1 and Rouge-2 scores between each sentence c in the set $S - S'$ and the reference summary r, where the reference summary r is typically the human-generated abstract summary provided with the article. The Rouge scores provide a measure of the similarity between the sentence and the reference summary, with higher scores indicating greater similarity.

As shown in Equation (1), the sentence with the highest Rouge score is then selected to be added to the set S', and the process is repeated until the desired number of summary sentences has been selected. By iteratively selecting the most similar sentences, the extractive summarization algorithm can generate a summary that accurately reflects the main ideas of the source document. Equation (1) shows that Rouge-1 and Rouge-2 metrics are used to select sentences to create a summary.

$$s' = s' \cup \underset{sen \in S - S'}{\operatorname{argmax}}(Rouge - 1(sen, r) + Rouge - 2(sen, r)) \quad (1)$$

Based on the metric in (1), three sentences are selected from the source document as the target summaries for the extractive summarization task (Nallapati, Zhai, & Zhou, 2017). Afterward, we create an array whose length is equal to the number of sentences in the source document, initialized to all zeros. Each number in the array corresponds to a sentence at the same index position, and the corresponding array value for the sentence that is selected as part of the extractive summary is marked as 1. This is about how to generate a target summary for the extractive summarization task. The following subsection describes how to apply pre-train language model encoders to encode the article and summary in the dataset.

Data Processing for Pre-train Language Model

After the text data has been tokenized and separated into the source document and extractive summary, we need to convert text data into a format that computers and NLP language models can recognize. In the CNN/DM dataset, there is a small portion of data where the source document is empty and only highlights exist. This indicates that the news topic may be an image or video and does not contain text. Additionally, some news articles in the dataset are too short. These data are not suitable for summarization tasks, so they need to be removed first. If the token count in the source text is less than 300, the text is deleted. Then, all sentences in the source text are restored to reconstruct the original document. Afterward, we process the text to remove sentences that are too long or too short. We iterate through each sentence in the text and remove sentences with fewer than 4 or more than 150 tokens. If a sentence is removed, the corresponding element in the summary sentence array is also deleted.

The next step is to encode the data. Encoding involves adding special tokens to the text data and transforming the words or characters in the text into vector space values. Liu (Liu, 2019) proposes an encoding strategy for input sequences in the pre-trained NLP neural network to support extractive summarization tasks. For the source document, the improved encoding rules insert special tokens [CLS] and [SEP] at the beginning and end of the sentence, respectively, to mark the start and end of the sentence. Interval paragraph embedding is used to distinguish sentences better, using different paragraph codes for odd or even numbered sentences. The position encoding uses position embedding to indicate the position of each token in the input sequence, allowing the model to capture the order of the tokens. The encoded text is then converted to an id sequence using the BERT or RoBERTa model Tokenizers (Devlin, Chang, Lee, & Toutanova, 2018; Liu, et al., 2019).

BERT and RoBERTa use different encoding methods. BERT uses character-level Byte-Pair Encoding (BPE) with a word list size of 30K, while RoBERTa uses byte-level BPE encoding (Wang, Cho, & Gu, 2019) with a word list size of 50K. The two encoding methods have a different granularity of subword units. The character-level BPE decomposes words into their constituent characters. For example, the word "word" may be broken down into subwords "w," "o," "r," and "d." The sequence of IDs generated in the BERT model for this word would be [1059, 1051, 1054, 1040]. On the other hand, bytes-level BPE breaks words into individual bytes and then combines them into larger subword units based on their frequency. This allows the model to handle a wider range of characters, including those not in the Latin alphabet, such as Chinese or Arabic characters. In the RoBERTa model, the encoding value of "word" is 14742. However, byte-level BPE may result in a larger vocabulary than character-level BPE because there are more byte-level combinations than character-level combinations. In our experiments, we observed the performance of the extractive summary created by our proposed neural network model based on the BERT and the RoBERTa models.

Data Processing for Knowledge Graph

In this NLP task, we aim to extract essential information from news articles in the CNN/DM dataset and build a knowledge graph. Knowledge graphs from news articles can significantly enhance the performance of the summarization algorithm. The knowledge graph serves as a reference that can be used to extract critical information from the text, thereby making the summary more concise and accurate. This task involves various steps from obtaining the raw news articles to constructing a knowledge graph. Firstly, we need to extract and organize the original news articles from the dataset. We directly segment the articles using the Stanford CoreNLP tool, which splits the character words and compound words in the article into different words. This can cause information loss or confusion in the articles. So, we need to collate original documents and extract the article parts directly from the dataset to maximize the content of the original news.

Once we have the articles, we need to extract the entities mentioned in the text using techniques such as Named Entity Recognition (NER) and coreference resolution. NER identifies and classifies named entities, such as people, organizations, and locations. Coreference resolution helps find all the expressions in the text that refer to the same entity, thus providing a complete view of the entities mentioned in the article. Once the entities have been identified, we can extract the relevant information and construct the triples (subject, predicate, and object) from the sentences in the text.

The subject and object of each triple represent the nodes in the knowledge graph, while the predicate represents the edges that connect these nodes. By constructing the knowledge graph, we can visualize the relationships between different entities and gain insights into the information contained within the text.

Text Collation

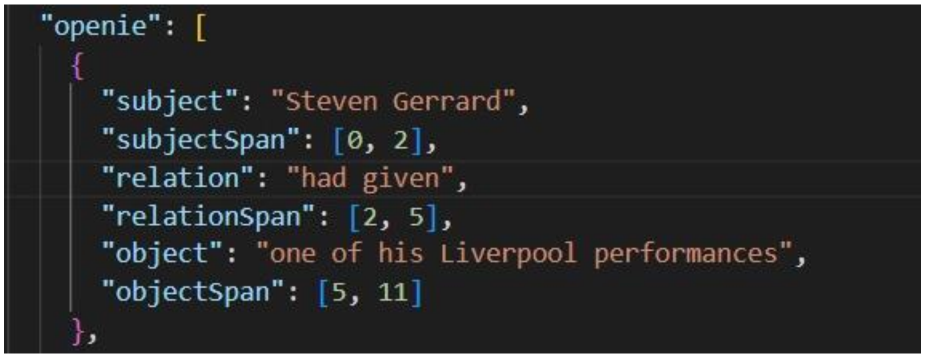
Text collation helps to generate higher-quality articles for the knowledge graph. We split the article and highlights and tokenize the article as input data to the pre-trained language model. However, it is a challenge to accurately tokenize the text, especially when dealing with hyphens, abbreviations, and compound words. This situation can lead to errors in tokenization, which is even more pronounced when reconstructing press releases from tokenized portions of the source text. Words with punctuation can cause problems in text reconstruction, resulting in the loss of valuable information. Therefore, news articles must be extracted directly from the original data, allowing us to maintain the integrity of punctuated words. Another challenge is that many sentences in the original text are divided into parts by the line break (i.e., '/n'). To accurately reconstruct these sentences, we adopted the strategy of reading the text in the target document line by line and removing all line breaks in the read text. By doing so, we can accurately reconstruct the split sentences and significantly reduce the loss of information in building the knowledge graph.

Coreference Resolution

When writing an article, pronouns, titles, and abbreviations are often used to refer to previously mentioned entities by their full name. This avoids the problem of repetitive use of the same word, which can make the sentence cumbersome. However, we need to convert the entire article into a knowledge graph when constructing a knowledge graph. From a global perspective, problems with ambiguous references are widespread. Different entities in the same document are replaced by the same pronoun, such as he, she, or it. Therefore, we need to perform coreference resolution to replace all pronouns with the named entities. We used spaCy3 (Neumann, King, Beltagy, & Ammar, 2019) and coreference toolkits with Python to implement coreference resolution. To perform coreference resolution, one first needs to load the en_core_web_trf component in spaCy3 and add the Coreferee pipeline to the language model. Then, the NLP model generated by spaCy3 is used to identify the pronouns or noun phrases and link them to the entities in the text. The entire process is accelerated using GPUs, and the entire processing workflow takes approximately 100 hours to complete. An example of coreference resolution is shown in Table 4. In this example, the pronunciation ‘he’ represents ‘Nicholas Levene’ at the beginning of the document. The conference resolution replaces ‘he’ with ‘Levene.’

Information Extraction for Triples

OpenIE (Angeli, Premkumar, & Manning, 2015) is a command in Stanford CoreNLP used to process all the sentences in a document and extract multiple triplets. During the processing, each triplet is assigned a probability value indicating whether the triplet is correct or not. By default, OpenIE sets the affinity probability to one-third, which means that a triple is output by OpenIE only when its probability of being correct is higher than one-third. To improve the accuracy of the information extraction results, we set the affinity probability cap to two-thirds. However, given that our dataset contains over 300,000 data, processing this dataset using the Stanford OpenIE command is slow and may lead to system crashes due to memory issues. Therefore, we set the thread number of OpenIE to 8 and allocated 32GB of memory using '-mx32g' to prevent command crashes. This approach greatly enhances the processing efficiency of the command and helps avoid unexpected errors. The extracted triplet is shown in Figure 1.



```
"openie": [
  {
    "subject": "Steven Gerrard",
    "subjectSpan": [0, 2],
    "relation": "had given",
    "relationSpan": [2, 5],
    "object": "one of his Liverpool performances",
    "objectSpan": [5, 11]
  },
```

Figure 1. A JSON format for Triple Data

The OpenIE data, stored in JSON format, in Figure 1 contains a triple that was generated from the original sentence. The triple consists of a subject, relation, and object, with the span of their index in the original sentence. In this case, the subject is "Steven Gerrard," and its span is from the first to the second word of the source sentence. The relation between the subject and the object is "had given," with a span of the index from the third to the fifth word of the sentence.

The object is "one of his Liverpool performances," and its span is from the fifth to the eleventh word of the original sentence. The data is a clear representation of how OpenIE can extract meaningful triples from a given sentence, providing insight into the relationships between different entities. With this information, we can analyze and interpret the text more comprehensively and efficiently.

Knowledge Graph Construction

Once the triples are extracted from the text, they must be combined to form a knowledge graph. The first step is to process the redundant triples. Table 4 presents four triples that come from the set of triples extracted from the first sentence of the news article in Table 1 by Stanford OpenIE. These triples consist of a subject, a relation, and an object, along with their corresponding index spans in the original sentence. The information extraction tool allows us to extract important information from sentences for the summarization task. However, we can observe that the first and fourth triples in the left half of Table 4 share an identical relationship and object parts and have a high degree of overlap in the subject part. This indicates the presence of redundant triples generated by Stanford OpenIE, which needs to be filtered out. To eliminate redundancy between the extracted triples, we select two triples at a time and calculate their degree of overlap to determine whether there is redundancy. The formula for determining whether redundancy exists is as follows:

$$Overlap(triple1, triple2) = \frac{Overlap_{subject} + Overlap_{relation} + Overlap_{object}}{\left(length_{triple1} + length_{triple2}\right)/2} \geq 0.75 \quad (2)$$

In Equation (2), $triple1\ and\ triple2$ are the selected triple, $Overlap_{subject}$, $Overlap_{relation}$, and $Overlap_{object}$ are the number of overlaps between the indexes of the span of the subject, relation, and object of the two triples, respectively. $length_{triple1}$ and $length_{triple2}$ are the total length of the sum of the spans of the three parts of the two triples. To calculate the overlap degree, $Overlap(triple1, triple2)$, we divide the sum of the overlap parts of the spans of the two triples by the average length of the spans of the two triples. If the value is larger than 0.75, we assume that redundancy exists between the two triples and delete one of them until there is no redundancy in the new triple set. The left part of Table 4 is the original set of triples, and the right part is the filtered set of triples. We can observe that the filtered triples do not have redundant relationships with each other. By filtering the triples, we can simplify the extracted triples, reduce the waste of unnecessary computational consumption, and make the generated knowledge graph more concise.

Table 4

Triples before and after Filter

Triples	Filtered Triples
{'subject': 'Nicholas Levene 48', 'subjectSpan': [0, 4], 'relation': 'was', 'relationSpan': [5, 6], 'object': 'jailed for 13 years November', 'objectSpan': [6, 12]},	{'subject': 'Nicholas Levene 48', 'relation': 'was', 'object': 'jailed for 13 years November'},
{'subject': 'Levene', 'subjectSpan': [13, 14], 'relation': 'orchestrating', 'relationSpan': [15, 16], 'object': 'luative Ponzi scheme', 'objectSpan': [17, 20]},	{'subject': 'Levene', 'relation': 'orchestrating', 'object': 'luative Ponzi scheme'},
{'subject': 'Nicholas Levene', 'subjectSpan': [0, 2], 'relation': 'was jailed at_time',	{'subject': 'Nicholas Levene', 'relation': 'was jailed at_time', 'object': 'November'}

(Table Continues)

Table 4, Continued

Triples	Filtered Triples

'relationSpan': [5, 7],
'object': 'November',
  'objectSpan': [11, 12]}]
…………
 	
{'subject': 'Nicholas Levene',
  'subjectSpan': [0, 2],
  'relation': 'was',
  'relationSpan': [5, 6],
  'object': 'jailed for 13 years last November',
  'objectSpan': [6, 12]},	

Table 4 shows that the filtered triples, while not redundant, may have slight differences in expressing the same entity. In the filtered triples in Table 4, 'Nicholas Levene 48,' 'Levene,' and 'Nicholas Levene' all serve as the subject of a triple and refer to the same person in the original text. When constructing the knowledge graph, three subjects with the same content are considered as three different nodes because of the subtle differences, which is not conducive for constructing the knowledge graph and reasoning about the knowledge structure of the articles. Therefore, we need to unify similar triple components. In the process of unifying triples, we compare each triple. If two triples have a 50% overlap in their subject, relation, or object part, we unify the overlapping components, and the smaller span of the two components is replaced by the larger one. For example, in Table 4, the subject part of the three triples has slight differences, and after unification, the subject part of all three triples is 'Nicholas Levene.'

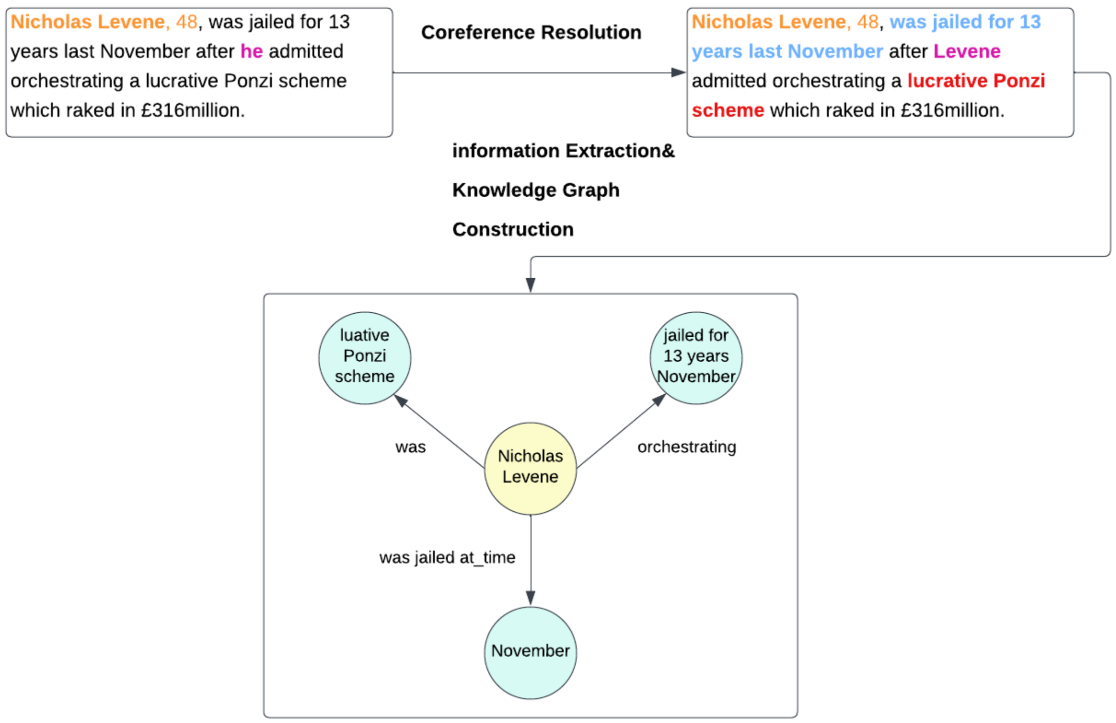

Nicholas Levene, 48, was jailed for 13 years last November after he admitted orchestrating a lucrative Ponzi scheme which raked in £316million.
Coreference Resolution
Nicholas Levene, 48, was jailed for 13 years last November after Levene admitted orchestrating a lucrative Ponzi scheme which raked in £316million.
information Extraction&
Knowledge Graph
Construction
luative Ponzi scheme
jailed for 13 years November
was
orchestrating
Nicholas Levene
was jailed at_time
November

Figure 2. An Example of Knowledge Graph

After filtering and unifying the triples, we need to build the knowledge graph. The nodes are the subject and object of the triple, and the edge is the triple's relation. Figure 2 is a visualized knowledge graph by the first sentence in Table 1. As shown in Figure 2, the knowledge graph is a directed graph in which the relationships between any connected pair of nodes are not bidirectional. The relationship exists only between the subject and the object and always points from subject to object.

Before training the knowledge graph, the triples need to be encoded. We use BERT and RoBERTa encoders to encode the subject, relationship, and object separately. Since some nodes in the graph are phrases, they need to be tokenized and then encoded as id sequences. Then the encoded triples are processed as a knowledge graph and saved into the dataset with the encoded source document and summary data for training and testing.

CHAPTER IV: METHODOLOGY

In this chapter, we describe our extractive summarization framework. In the task of extractive summarization, the system is required to extract the most representative sentences from the document and create a summary. Our work uses a neural network model to predict which sentences belong to the summary. To train and fit the model, we need to identify which sentences in the source document belong to the summary and select these sentences as output. The input of our model consists of the token of the input document $X = \{x_k\}$ and knowledge graph $G = \{(V, E)\}$, where the V is the node and the E is the edge set. The output of the model $S = \{s_n\}$ is the score of each sentence. We choose the top three highest score sentences as the summary. In deep learning, we process the extractive summary as a sentence classification problem (Liu, 2019) to classify whether the sentence belongs to the summary. We extract data from the source document and construct a knowledge graph to represent its structure and infer further information. We employ the source document and the knowledge graph as inputs to the neural network. Our model architecture is shown in Figure 3. In our approach, as shown in Figure 3, the neural network consists of (1) an encoder for the source document and a knowledge graph, (2) a classifier for sentence classification prediction, and (3) other components (such as a transformer). In Figure 4, we present our model with more details about each component.

Encoder

To tackle extractive text summarization, we use a neural network involving two inputs: the tokenization source document and the knowledge graph from the source document. Our proposed architecture in Figure 3 consists of two parts: an encoder part and a classifier part. The encoder part consists of a graph encoder for the knowledge graph and a source document encoder like BERT (Devlin, Chang, Lee, & Toutanova, 2018) and Roberta (Liu, et al., 2019).

The knowledge graph encoder converts a knowledge graph into a feature vector that the subsequent layers can use. The source document encoder processes the tokenized source document and converts it into a sequence of feature vectors.

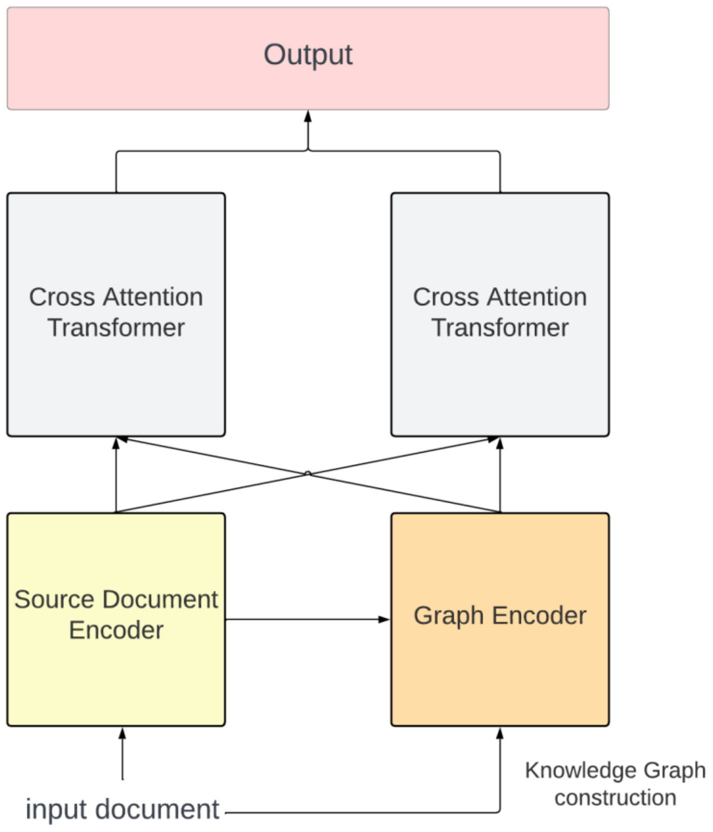

Output
Cross Attention Transformer
Cross Attention Transformer
Source Document Encoder
Graph Encoder
input document
Knowledge Graph construction

Figure 3. The Proposed Model Architecture

Source Document Encoder

We begin by tokenizing the document and feeding the input source document *x* to either the BERT (Devlin, Chang, Lee, & Toutanova, 2018) or RoBERTa (Liu, et al., 2019) model. The output of BERT/RoBERTa is the embedding document. We then employ a two-transformer layer, x2 Transformer (Liu, 2019), shown in Figure 4, to process the document-level feature representation to boost the performance of the extractive summarization algorithm:

$$\tilde{x}^l = \left(x^{l-1} + MHAttention(x^{l-1})\right) \tag{3}$$

$$x^l = LN(\tilde{x}^l + FFN(\tilde{x}^l)) \quad (4)$$

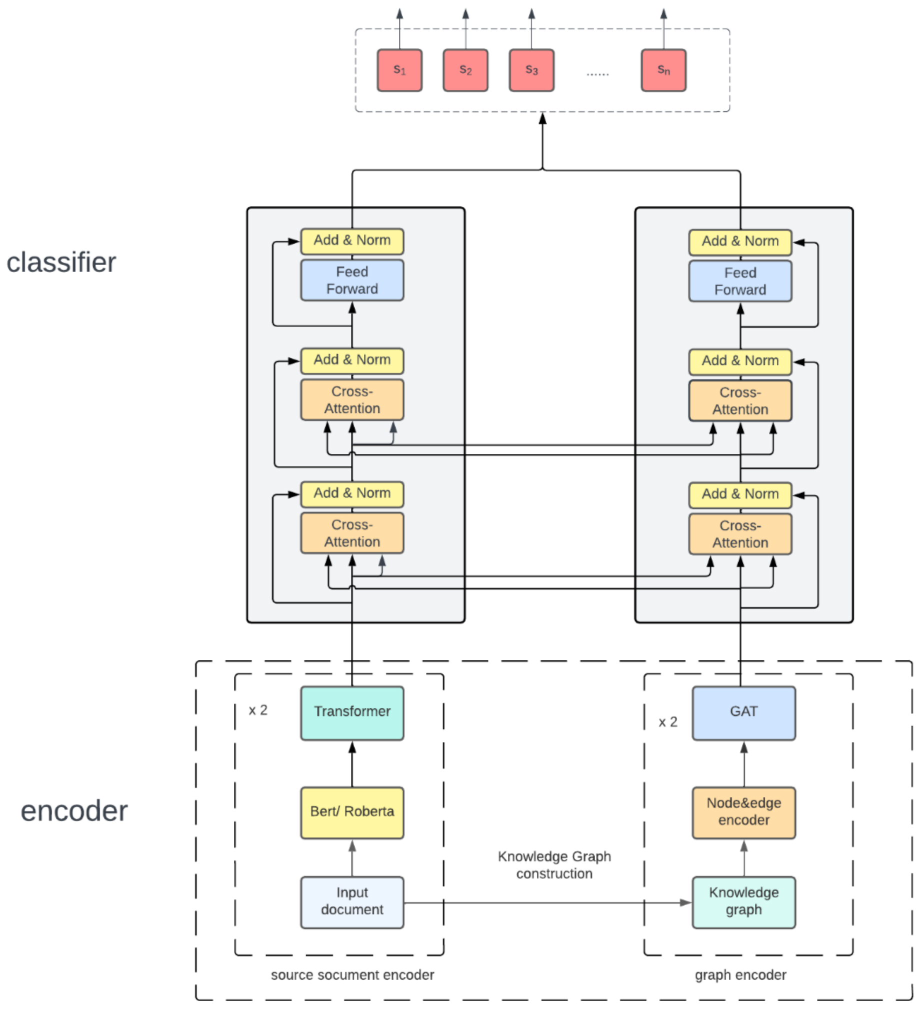


Figure 4. The Proposed Model Details

Where the output x of the BERT/RoBERTa model is the input of the transformer. $MHAttention$ in (3) is a multi-head attention operation (Vaswani, et al., 2017). Multi-head attention allows the model to capture more complex information from multiple perspectives from the input data, leading to more accurate and robust results. FFN in (4) is the feed-forward operation, and LN in (4) is the layer normalization operation. The output of the transformer layer is sent back into the cross-attention transformer layer in the classifier.

Graph Encoder

The first step of the graph encoder is to construct the knowledge graph by triples extracted from the source document by Stanford OpenIE. The knowledge graph is created by representing the subject and object entities as nodes and the relations between them are represented as edges pointing from a subject to an object. The next step involves encoding each node and edge in the knowledge graph using the corresponding encoder from the BERT or RoBERTa models. Additionally, we need to ensure that the source document encoder and the graph encoder are integrated to generate a unified representation. Therefore, we encode the source document and the knowledge graph using the same encoder. After the graph encoding step, a two-layer graph attention network (GAT) is used to process the encoded graph (Veličković, et al., 2017). In our experiments, the two-layer GAT network shares their parameters, and the GAT network is iterated three times in one training to fit the complex parameters in GAT. The output of GAT is transformed into a sequence with the same dimensionality as the output of the language model as the BERT/RoBERTa model in the source document encoder.

Classifier

Our classifier utilizes two parallel cross-attention transformer architectures and a linear output layer. The output from the transformer in the source document encoder and the GAT component in the graph encoder is fed into separate cross-attention transformers. The information from the source document encoder and the graph encoder is mutually encoded in both cross-attention transformers. This allows the two cross-attention transformers to encode information from the source document and the knowledge graph. Finally, a linear output layer is used to classify which sentences should be included in the summary.

Cross Attention

In our model, we incorporate cross-attention to enable the interaction between the inputs from the transformer in the source document encoder and GAT in the graph encoder. Specifically, when processing the outputs from the transformer in the source document encoder, we introduce the output sequence from GAT as weights to generate the next hidden state in the left-side cross-attention transformer in Figure 4. The output from the transformer in the source document encoder is also the weight of the right-side cross-attention transformer in Figure 4. The output of GAT is the input to the right-side cross-attention transformer. By doing so, the model can selectively attend to relevant information from both the transformer and GAT, thus enabling better information exchange between the two types of data. The output of the cross-attention layer in Figure 4 is calculated as follows:

$$\alpha_1 = \frac{\exp(h_2 h_1^T)}{\sum \exp(h_2 h_1^T)} \tag{5}$$

$$\bar{h}_1 = \alpha_1 h_1 \tag{6}$$

$$\alpha_2 = \frac{\exp(h_1 h_2^T)}{\sum \exp(h_1 h_2^T)} \tag{7}$$

$$\bar{h}_2 = \alpha_2 h_2 \tag{8}$$

When calculating the output of cross-attention, we use the input from the transformer and GAT part to get the weight of each part. The h_1 and h_2 are the input from the transformers in the source document encoder and the GAT of the graph encoder, respectively. The α_1 and α_2 are the weights of h_1 and h_2, respectively. The $\bar{h}_1$ and $\bar{h}_2$ are the output of the cross-attention layer whose output is added to input h_1 and h_2 and then fed into the normalization layer at the next step of cross-attention layer. The cross-attention mechanism allows the neural network to learn contextual relationships by analyzing the information from the transformer output in the source document decoder with article structure information from the graph encoder. Meanwhile, the transformer information from the source document encoder provides additional information to the neural network when analyzing the text structure information from the graph attention network.

Output

To classifier the sentences, the outputs of the two transformer layers are concatenated into one output. The output is processed using a full connection layer and applying the sigmoid function as activate function (Dubey, Singh, & Chaudhuri, 2022) to get the sentence score:

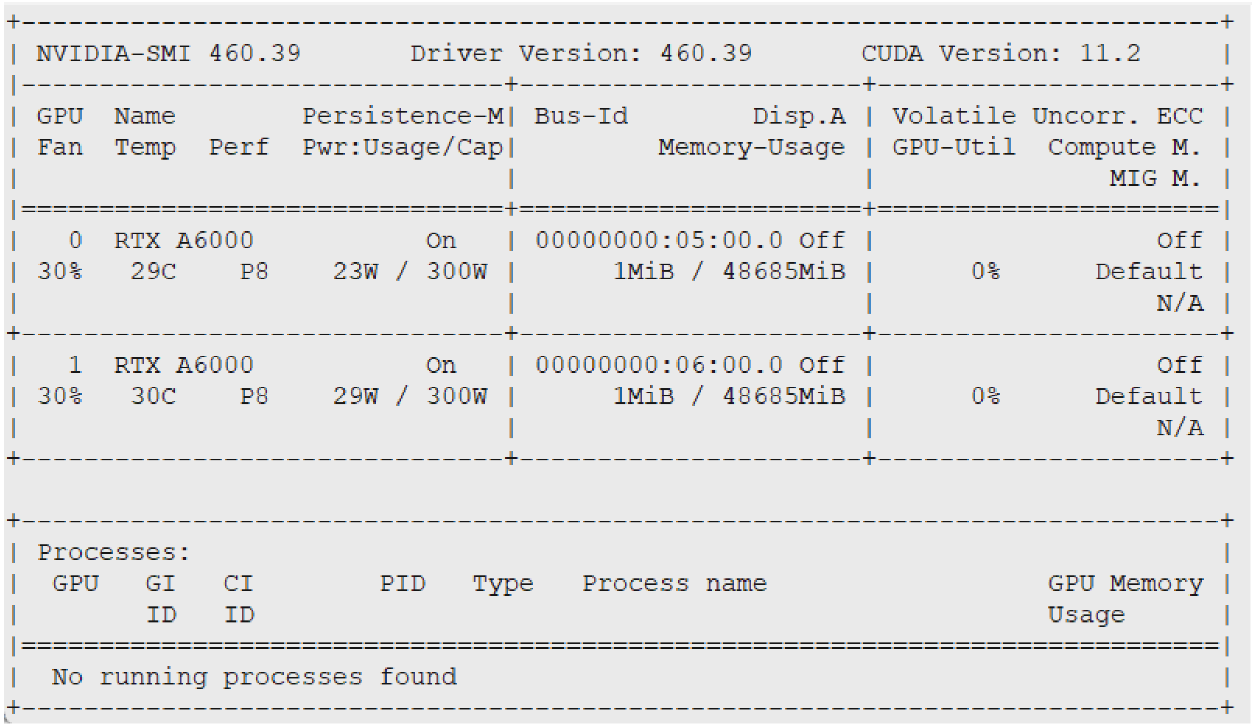
$$Y = sigmoid(W[h_1, h_2] + b) \tag{9}$$

In Equation (9), the $[h_1, h_2]$ is the concatenated input from two outputs of the cross-attention transformer modules. Y is the output of our model, which represents the probability that each sentence belongs to the summary sentence. Cross-entropy loss is applied as the loss function for training and validation. When we evaluate the performance of the trained model on the test set, we select the top-three sentences based on the probability Y in the source document as the generated summary.

CHAPTER V: EXPERIMENT

Experiment Environment

In this experiment, the Lambda server provided by Illinois State University was used for processing and analyzing natural language data. The server is equipped with two RTX A6000 GPUs, each with a memory capacity of 48685MB, and is powered by the NVIDIA-SMI driver version 460.39 and CUDA version 11.1. The GPUs were utilized for deep learning tasks such as training and testing of neural networks.



```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.39       Driver Version: 460.39       CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  RTX A6000           On   | 00000000:05:00.0 Off |                  Off |
| 30%   29C    P8    23W / 300W |      1MiB / 48685MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  RTX A6000           On   | 00000000:06:00.0 Off |                  Off |
| 30%   30C    P8    29W / 300W |      1MiB / 48685MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Figure 5. Lambda Server at Illinois State University

Our experiment utilized the PyTorch framework (Paszke, et al., 2019) to build our designed network. The Transformer Toolkit (Wolf, et al., 2019) was used in constructing the BERT and RoBERTa models. The Transformers is an open-source software library developed by Hugging Face (Wolf, et al., 2019) that provides a powerful set of tools for processing Transformer models, such as BERT and RoBERTa. The Transformer model allows researchers to fine-tune pre-trained Transformer models for specific NLP tasks such as text summarization. PyTorch can work together with the Transformers, and the latter provides an easy-to-use API and pre-trained models to support NLP tasks.

To build a graph attention network on the PyTorch framework to analyze our knowledge graph, we use the PyG toolkit (Fey & Lenssen, 2019). PyG is an open-source Python library for deep learning on graph data. It is built upon PyTorch and can easily design and train graph neural networks by various blocks provided by PyG.

Models have been trained with Adam optimization with learning rate, $\beta_{1,}$ and β_2, which are set as 0.001, 0.9, and 0.999, respectively, in training on GPUs on the Lambda server. The training step is set as 200,000, and the batch size is set as 3000. The golden summary generated by the greedy algorithm is used to train our model. Model checkpoints were saved every 2000 epochs and evaluated on the validation set. The best three checkpoints were selected based on the loss value on the validation set.

Based on the selected checkpoint neural network models, we use the selected models to generate a summary per document and test the summary quality on the test set. The first step is to use the model to score each sentence of the input document. The sentences were ranked according to their highest to lowest scores. The top-ranked three sentences with the highest scores are selected and used as machine-generated summaries. Rouge is used as the evaluation metric when evaluating the extractive summarization model performance. The average rouge value of three checkpoint models is reported on the test set.

Evaluation Metric

Rouge (Recall-Oriented Understudy for Gisting Evaluation) (Lin, 2004) is an evaluation metric used in NLP text to evaluate the quality of a summary by measuring the overlap between the model-generated summary and the reference summary. In our experiment, we applied three different variants, Rouge-1, Rouge-2, and Rouge-L, to evaluate the performance of the summaries. To get Rouge-1 and Rouge-2 values, we need the Rouge-N function. Rouge-N is calculated as follows:

$$Rouge - N = \frac{\sum_{S \in \{ReferenceSummaries\}} \sum_{gram_N \in S} Count_{match}(gram_N)}{\sum_{S \in \{ReferenceSummaries\}} \sum_{gram_N \in S} Count(gram_N)} \tag{10}$$

In obtaining the Rouge-N value, the first step is to tokenize the model-generated summary and the reference summary based on the size N in (10) Then calculate the number of matching n-grams (contiguous sequences of n words) between the reference summary and the model-generated summary and divide it by the total number of n-grams in the reference summary.

The Rouge-L is calculated as follows:

$$Rouge - L = \frac{LCS(C,\ S)}{len(S)} \tag{11}$$

Rouge-L is used to calculate the longest common subsequence (LCS). To get Rouge-L, we first compute the LCS of words in the machine-generated summary C and the reference summary S. Then we calculate the length, $len(S)$, of the reference summary S. We compute the Rouge-L score by dividing the LCS length by the reference summary length.

Pyrouge is a Python library that provides convenient interfaces for Rouge evaluation in text summarization. Pyrouge can be used with any plain text format summarizer to compute the Rouge-1, Rouge-2, and Rouge-L metric scores between the generated summaries and the reference summaries. In our experiment, we facilitate the Pyrouge library to get the Rouge-1, Rouge-2, and Rouge-L metric scores for testing.

CHAPTER VI: RESULTS

To evaluate our approach, we compare our neural network with other existing extractive summarization approaches and report unigram and bigram overlap metrics using Rouge-1 and Rouge-2 and the longest common subsequence Rouge-L. The results are shown in Table 5. To evaluate our approach, we compare our neural network model with other extractive summarization approaches. The results are shown in Table 5. The first section contains LEAD and ORACLE. LEAD is an extractive baseline that uses the first three sentences of the document as a summary. ORACLE calculates the best possible summary that can be generated by selecting sentences from the source document by a greedy algorithm. ORACLE provides an upper bound on the performance that any extractive summarization techniques can work as a guide.

Table 5

Results on CNN/DM Test Dataset

Model	Rouge-1	Rouge-2	Rouge-L
LEAD	40.43	17.32	36.67
ORACLE	52.59	31.23	48.47
NEUSUM	41.59	19.01	37.98
BERTSUM	43.23	20.22	39.60
BertSumExt	43.85	20.34	39.90
MATCHSUM (BERT-base)	44.22	20.62	40.38
MATCHSUM (RoBERTa-base)	44.41	20.86	40.55
BertSumExt (our implementation)	43.44	20.38	39.77
MATCHSUM (RoBERTa-base) (our implementation)	43.87	20.32	39.95

(Table Continues)

Table 5, Continued

Model	Rouge-1	Rouge-2	Rouge-L
Our model (BERT-base) – with a knowledge graph	43.95	20.45	39.97
Our model (RoBERTa-base) – without a knowledge graph	43.75	20.29	39.87
Our model (RoBERTa-base) – with a knowledge graph	44.11	20.61	40.29

The second part of Table 5 lists the performance of existing extractive deep-learning models, which include NEUSUM (Zhou, et al., 2018), two SOTA models BERTSUM (Liu, 2019) and MATCHSUM (Zhong, 2020), with the BERTSUMEXT model which replaces the BERT encoder with the RoBERTa encoder. The MATCHSUM model has two versions: BERT-based and RoBERTa-base version. The performance of the MATCHSUM RoBERTa-base model is better than the MATCHSUM BERT-base model. All the results in the second part of Table 5 come from the corresponding extractive summarization systems (Zhou, et al., 2018; Liu, 2019; Zhong, et al., 2020).

The last part of Table 5 shows the performance of various models in our experiments. In our experiments, we train and test our model in the Lambda server at Illinois State University. To prevent interference from other factors, we re-train and implement the BERTSUMEXT and MATCHSUM in the same experiment environment and setting as our model experiment environment and setting. Our model has three variants: a model with a BERT-base encoder with the knowledge graph, a model with a RoBERTa-base encoder with the knowledge graph, and a model with a RoBERTa-base encoder without the knowledge graph. All three variants of our model outperformed our implementation of the BERTSUMEXT model. Our model enhanced by the knowledge graph performs better than our implementation of MATCHSUM. Compared to MATCHSUM and BERTSUMEXT, using a knowledge graph to provide a structured representation of entities and relationships from source documents could better understand the context of the document to boost the performance of the extractive summarization task. We also trained the model without the knowledge graph and the corresponding model structure and compared this model with our original model. The performance of our model with the knowledge graph is much better than our model without the knowledge graph. The RoBERTa-based model with knowledge graph reached the best result, which is 44.11, 20.61, and 40.29 in R-1, R-2, and R-L, respectively, which proved that our model using knowledge graphs outperforms other existing models as shown in Table 5.

CHAPTER VII: CONCLUSION

In this thesis, we proposed a neural network model with BERT/RoBERTa for contextual understanding and with a knowledge graph for document structure reasoning. The relevant information extracted from the source text and knowledge graph was inter-encoded. We encoded the source document and all elements in the knowledge graph and used pre-trained language models and a graph neural network to extract the important sentences from the source document. Then we applied a cross-attention mechanism for interceding information in both data from the source document encoder and graph encoder. In terms of the performance on the CNN/DM dataset, our model achieves the best result compared with other existing extractive summarization work, which proves that we use a cross-attention mechanism to cross-encode features extracted from the knowledge graph with features extracted from the source document, bringing text structure information to the extractive summarization task and improving the quality of extractive summarization.

In the future, we will continue to adjust the cross-attention network component and design the corresponding network structures for the output data from the source document encoder and the graph encoder, respectively, to exploit the potential of the text summarization model we developed. We will also focus on abstractive summarization. The task of abstractive summarization is to generate a new summary based on the source document, so we need to develop a new decoder for our text summarization model.

REFERENCES

Angeli, G., Premkumar, M. J., & Manning, C. D. (2015). Leveraging linguistic structure for open domain information extraction. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 344-354.

Belwal, R. C., Rai, S., & Gupta, A. (2021). A new graph-based extractive text summarization using keywords or topic modeling. *Journal of Ambient Intelligence and Humanized Computing*, *12*(10), 8975-8990.

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*.

Dubey, S. R., Singh, S. K., & Chaudhuri, B. B. (2022). Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*.

Erkan, G., & Radev, D. R. (2004). Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of artificial intelligence research, 22,* 457-479.

Fey, M., & Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428*.

Garg, N., Favre, B., Reidhammer, K., & Hakkani Tür, D. (2009). *Clusterrank: a graph based method for meeting summarization.* No. REP_WORK: Idiap.

Gokhan, T., Smith, P., & Lee, M. (2022, October). GUSUM: Graph-Based Unsupervised Summarization using Sentence Features Scoring and Sentence-BERT. *Proceedings of TextGraphs-16: Graph-based Methods for Natural Language Processing*, (pp. 44-53).

Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., & Blunsom, P. (2015). Teaching machines to read and comprehend. *Advances in neural information processing systems, 28*.

Huang, L., Wu, L., & Wang, L. (2020). Knowledge graph-augmented abstractive summarization with semantic-driven cloze reward. *arXiv preprint arXiv:2005.01159*.

Jin, H., Wang, T., & Wan, X. (2020, July). Multi-granularity interaction network for extractive and abstractive multi-document summarization. *Proceedings of the 58th annual meeting of the association for computational linguistics*, (pp. 6244-6254).

Lin, C. Y. (2004, July). Rouge: A package for automatic evaluation of summaries. *Text summarization branches out*, (pp. 74-81).

Lin, C. Y., & Och, F. (2004). Looking for a few good metrics: ROUGE and its evaluation. *Ntcir workshop.*

Liu, Y. (2019). Fine-tune BERT for extractive summarization. *arXiv preprint arXiv:1903.10318*.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., & Stoyanov, V. (2019). *Roberta: A robustly optimized bert pretraining approach*. Retrieved from arXiv: https://arxiv.org/abs/1907.11692

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J. R., Bethard, S., & McClosky, D. (2014, June). The Stanford CoreNLP natural language processing toolkit. *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, (pp. 55-60).

Mihalcea, R., & Tarau, P. (20004). Textrank: Bringing order into text. *Proceedings of the 2004 conference on empirical methods in natural language processing*, (pp. 404-411).

Nallapati, R., Zhai, F., & Zhou, B. (2017). Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. *Proceedings of the AAAI conference on artificial intelligence*, (pp. 3075-3081).

Nallapati, R., Zhou, B., dos Santos, C., Gủlçehre, ç., & Xiang, B. (2016). Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond. *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, (pp. 280-290).

Narayan, S., Cohen, S. B., & Lapata, M. (2018). Ranking sentences for extractive summarization with reinforcement learning. *arXiv preprint arXiv:1802.08636*.

Neumann, M., King, D., Beltagy, I., & Ammar, W. (2019). ScispaCy: fast and robust models for biomedical natural language processing. *arXiv preprint arXiv:1902.07669*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., . . . Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems, 32*.

Tan, J., Wan, X., & Xiao, J. (2017). Abstractive document summarization with a graph-based attentional neural model. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, (pp. 1171-1181).

Tang, P., Hu, K., Yan, R., Zhang, L., Gao, J., & Wang, Z. (2022). OTExtSum: Extractive Text Summarisation with Optimal Transport. *arXiv preprint arXiv:2204.10086*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., . . . Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems, 30*.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Vhatkar, A., Bhattacharyya, P., & Arya, K. (2020). Knowledge graph and deep neural network for extractive text summarization by utilizing triples. *Proceedings of the 1st Joint Workshop on Financial Narrative Processing and MultiLing Financial Summarisation*, (pp. 130--136).

Wang, C., Cho, K., & Gu, J. (2019). Neural Machine Translation with Byte-Level Subwords. *arXiv preprint arXiv:1909.03341*.

Wang, D., Liu, P., Zheng, Y., Qiu, X., & Huang, X. (2020). Heterogeneous graph neural networks for extractive document summarization. *arXiv preprint arXiv:2004.12393*.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., . . . Rush, A. M. (2019). HuggingFace's Transformers: State-of-the-art Natural Language Processing. *arXiv preprint arXiv:1910.03771*.

Wu, Z., Koncel-Kedziorski, R., Ostendorf, M., & Hajishirzi, H. (2020). Extracting summary knowledge graphs from long documents. *arXiv preprint arXiv:2009.09162*.

Yuan, R., Wang, Z., & Li, W. (2020). Fact-level extractive summarization with hierarchical graph mask on BERT. *arXiv preprint arXiv:2011.09739*.

Zhong, M., Liu, P., Chen, Y., Wang, D., Qiu, X., & Huang, X. (2020). Extractive summarization as text matching. *arXiv preprint arXiv:2004.08795*.

Zhou, Q., Wei, F., & Zhou, M. (2020). At which level should we extract? An empirical analysis on extractive document summarization. *arXiv preprint arXiv:2004.02664*.

Zhou, Q., Yang, N., Wei, F., Huang, S., Zhou, M., & Zhao, T. (2018). Neural document summarization by jointly learning to score and select sentences. *arXiv preprint arXiv:1807.02305*.

Zhu, C., Hinthorn, W., Xu, R., Zeng, Q., Zeng, M., Huang, X., & Jiang, M. (2020). Enhancing factual consistency of abstractive summarization. *arXiv preprint arXiv:2003.08612*.