Winter 1-31-2022

Extractive Text Summarization on Single Documents Using Deep Learning

Shehab Mostafa Abdel-Salam Mohamed
shehab_m@aucegypt.edu

The American University in Cairo

School of Sciences and Engineering

Extractive Text Summarization on Single Documents Using Deep Learning

A thesis submitted to the

Department of Computer Science and Engineering

In partial fulfillment of the requirements for the degree of

Master of Computer Science

By Shehab Mostafa Abdel-Salam Mohamed

B.Sc., Computer Engineering

The American University in Cairo

Under the supervision of

Prof. Ahmed Rafea

January / 2022

Abstract

The task of summarization can be categorized into two methods, extractive and abstractive summarization. Extractive approach selects highly meaningful sentences to form a summary while the abstractive approach interprets the original document and generates the summary in its own words. The task of generating a summary, whether extractive or abstractive, has been studied with different approaches such as statistical-based, graph-based, and deep-learning based approaches. Deep learning has achieved promising performance in comparison with the classical approaches and with the evolution of neural networks such as the attention network or commonly known as the Transformer architecture, there are potential areas for improving the summarization approach. The introduction of transformers and its encoder model BERT, has created advancement in the performance of many downstream tasks in NLP, including the summarization task. The objective of this thesis is to study the performance of deep learning-based models on text summarization through a series of experiments, and propose *"SqueezeBERTSum"*, a trained summarization model fine-tuned with the SqueezeBERT encoder which achieved competitive ROUGE scores retaining original BERT model's performance by 98% with ~49% fewer trainable parameters.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

NLP	Natural Language Processing
ATS	Automatic Text Summarization
SoTA	State of The Art
ML	Machine Learning
DL	Deep Learning
NER	Named Entity Recognition
POS	Part-Of-Speech Tagging
NN	Neural Network
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
FFN	Feed-Forward Network
MHA	Multi-Head Attention
LSTM	Long Short Term Memory
biLSTM	Bidirectional Long Short Term Memory
GRU	Gated Recurrent Unit
TF-IDF	Term Frequency - Inverse Document Frequency
CBOW	Continuous Bag-of-Words
GloVe	Global Vectors for Word Representation
Word2Vec	Word-To-Vector
BERT	Bidirectional Encoder Representations from Transformers
ELMo	Embeddings from Language Models
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
CB	Contextual-Bandit
TM	Text-Matching
PFC	Position-wise Fully Connected
CV	Computer Vision
Average_R	Average Recall (R) Score
Average_P	Average Precision (P) Score
Average_F	Average F-Score

Chapter 1: Introduction

This chapter provides an overview of the problem and the research area focused on solving it. It describes the opportunities when solving the problem and why it is essential to have an existing summarization solution. Afterwards, research challenges and objectives are presented in detail and followed with the thesis layout.

1.1 Problem Definition

The world of texts, instant document sharing, and constant growth of data, has grown phenomenally and still does, given the portability and the ease of using a mobile device to share and receive data on the fly. Now there is large amounts of data being shared among different mobile devices, and sometimes it carries redundant information or perhaps large data where a small segment can represent the main information of that data. Based on a study sponsored by EMC titled “Idc digital universe study: Big data, bigger digital shadows and biggest growth in the far east.”, the study estimates that by 2020 around 1.7 megabytes of new information will be created every second for every human being on earth which presents a dire need to have a software that can summarize a large document and provide a condensed version that still captures the information of that document. Text summarization is the process of extracting important sentences or snippets of a large document and combining them into a condensed version.

Text summarization (TS) is one of the most powerful research areas that has been in constant development. Text summarization is the process of extracting important sentences or snippets of a large document and combining them into a condensed version. Summarizing a text can be both time-efficient and cost-efficient. Time-efficiency would be the human interpreter reading a summary that grasps the main essence of a very large document instead of spending more time reading the whole document. Newsgroups can use a document summarization tool on several documents to gather the important information of each document that discusses the same topic. Cost-efficiency would be compressing the amount of data being transferred from a device to a device. Users who frequently check their phones prefer to receive their information as short as possible without staying connected to the internet for a long time to download the whole textual data. It would be cost-efficient if the user can have the choice of reading a summary before requesting the whole document to read. With todays’ rate of growth, it will soon become a necessity to have a tool that “digs” through multi-large documents and generates a quick insight of the data as a service to the human reader.

1.2 Motivation

Building a software that summarizes text is not a new idea introduced in the late century, many researchers have developed different versions of that software and tried to achieve the best accuracy which is creating a summary that would capture the main essence of the document. Text summarization became a research interest recently because today's machines are now equipped with Graphical Processing Units, or in short, GPUs which in result, gave more computational power for experimentations with different deep learning models to solve a classification/regression problem. The first Automatic Text Summarization was created by Luhn in 1958 based on term frequency [2]. The definition of a summary differs from one human reader to another, so each interpreter can generate a different condensed version of a document based on his/her belief of which are the most important or relevant sentences that need to be added to the final summary. Therefore, a text summarizer has to be evaluated with a certain metric which will be discussed later in this paper. A summary generated from a document might serve different purposes depending on the human reader. A reader might want to read a summary that gives an insight about the document before reading the entire document. Another reader might want to read the relevant keywords or tagged topics in this document without reading it. Before building a summarization software, there should be a formal definition of what the summary is and what the output of that software should be. The motivation behind this thesis is to explore different deep learning models for text summarization and produce a study on evaluating single-document summarization performance in terms of accuracy and training time.

1.3 Challenges

Challenges of performing extractive summarization tasks vary from selection of an evaluation metric to measure the performance of the summarizer, to selection of the type of document to be summarized. Generating a summary does not have an absolute correctness as different readers can produce different versions of a summary that can deliver the same value. Given this statement, researchers have widely adopted evaluation metric ROUGE to compare the overlap between the computer generated summary with the human generated summary or what is labeled as golden summary, this research challenge is elaborated in more detail in Part 2. The other research challenge is domain-specific summarization, which is the process of summarizing a special type or format of data from medical documents to an event-driven document such as story-telling documents, to academic documents. Those are different forms of data that can be summarized and might require a custom deep learning approach that needs to learn the structure of the data in order to produce a summary without missing important details. This challenge posed a requirement that any model training on a specific domain might produce good summaries on that type of data but may yield degrading performance on summarizing documents of different domains.

1.4 Goal & Objectives

The goal of this thesis is to identify performance improvement in extractive text summarization and evaluate an extractive approach that provides better performance either in accuracy increase or reduction of training time. Based on the stated goal, the objective of this thesis is to provide an in-depth study of BERT variants (also known as contextualized-word embeddings) and its performance with different deep learning models. In this objective, performance here can be divided into two metrics; accuracy and training time. Evaluating a BERT variant with a deep learning model will be through measuring the summary generation accuracy and the time taken to train that model. The summary generation accuracy will be evaluated using ROUGE scores. Throughout this thesis, the main objective will be answering two research questions which are the following:

1) How can a deep learning model improve summarization performance?
2) Which variant of the BERT model produces better summarization performance?

Answering these two research questions will be through a set of milestones. For research question (1), the first experiment is to reproduce the results obtained on the extractive text summarization to compare the improvements of three different neural architectures on the CNN/DM dataset using the bert-base variant. The three different architectures are; recurrent neural network, transformer network, and a fully-connected network as a classifier. The second research question is to find out which BERT variant produces the best summarization results. For research question (2), the methodology will be to conduct a series of experiments on fine-tuning a deep-learning model with a BERT variant and measuring both performance accuracies and training time. This methodology will also involve using Google Colab Notebooks to clone the code repository with the modified changes and utilize the available free GPU to train the models and report the findings through a demo Notebook.

1.5 Thesis Layout

The rest of this thesis is organized as follows: **Chapter 2** builds the core background of summarization, types of it, and the existing datasets that could be used to train an extractive text summarization. **Chapter 3** surveys related work on the extractive text summarization using supervised learning or unsupervised learning. **Chapter 4** describes the methodology through a series of experiments that will attempt to answer the two research questions mentioned in section 1.4. **Chapter 5** concludes the thesis with its findings and discusses future work that can be applied to the current contribution.

Chapter 2: Literature Review

This chapter discusses the related work in solving the text summarization problem. First, it covers the summarization definition and its types, and then it lists down the different approaches in text summarization. Second, this chapter discusses the literature of each approach-based summarization and identifies which dataset was used, which word embedding technique was applied, and which evaluation metric was used.

2.1 Summarization Definition

The task of automatic text summarization is composed of the following components:

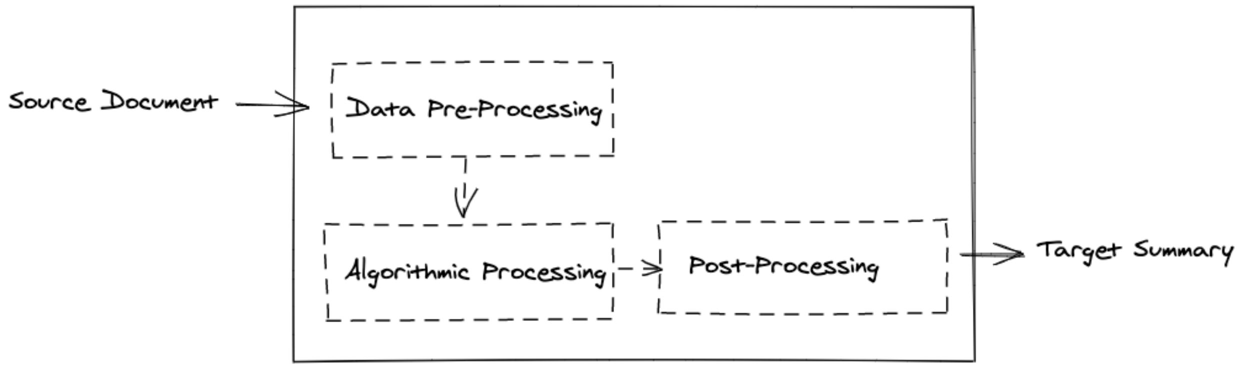


Figure 1: Overview of Automatic Text Summarization

1. Data pre-processing phase, the process of applying cleaning and transformation of the raw source document. Examples of data pre-processing techniques are; (1) noise removal which is the removal of any data that does not add value to the information in the document or summary. (2) sentence tokenization that splits the data into a set of sentences. (3) removal of punctuation marks. (4) work tokenization to split the sentence into a set of words or tokens in that matter. (5) removal of stop words to remove frequently occurring words such as (a), (an), (the), et cetera. (6) word stemming which is the removal of suffixes and prefixes. (7) lemmatization which is transforming the word to its base or dictionary word such as transforming the word [playing, played, plays] to its base form [play]. (8) part-of-speech tagging or PoS tagging in short, which is the process of assigning grammatical categories to a given word or token.

2. Algorithmic processing phase, it is covered more in the literature review on various ways of applying text summarization. Briefly, it is the phase related to applying an algorithmic approach

on generating a summary from the input pre-processed document. The approach could be either statistical, graph-based, topic-based, or machine learning-based. Algorithmic processing phase also includes either extractive or abstractive summarization which will be further discussed in part 2.1, extractive techniques are favored than abstractive techniques since the former has shown better performance and is relatively easier to implement [1].

3. Post-processing phase, the process of applying any necessary data transformation to the output of the algorithm processing phase to be given as a target summary. This phase can be optional to some approaches as summaries could be generated out from an existing approach without the need to apply any NLP techniques on it.

Summarization can be defined into two forms:

1. Types of Summary (Abstractive & Extractive)
2. Types of Information (Informative & Indicative)

For Summary type, text summarization can be split into two distinct approaches: extractive summarization and abstractive summarization. Extractive Summarization is a summary formed by reusing portions of the original document like words or sentences and gathering them in a summary. In an extractive approach, each sentence is ranked based on the most salient information and then ordered together into a summary while preserving the grammatical rules. SumaRuNNer [3] is one of several studies that performs extractive summarization. Abstractive Summarization is a summary formed by interpreting the original document and generating meaningful sentences in a shorter version. This approach involves more semantic understanding of the document by the software in order to write a summary in a human understandable form. An example of such an approach is "*Summarist*" which has modules that perform that type of summarization [4].

A summary does not have an absolute validation, it is the result of the reader's interpretation of a document based on his/her understanding. A valid summary can give a few important keywords and another valid summary can present a paragraph. A summary is split into two different groups of information:

1. Indicative summary which gives the main idea of the text to the reader. The length of this type of summarization is around 5-10% of the original document [5]. A summary like this is used to give readers a quick look at the headline and subtitles. The information on the back of movie packs is one example of an indicative summary.
2. Informative summary however, gives brief concise information of the original document. The length of the informative summary is around 15-20% of the original document [5] and can be a good replacement of the original document.

These are different approaches which could be used to perform an extractive summarization task. Using frequency count approach by **Luhn** [2]. Using a graph-based approach by **TextRank** [12] that represents a document as a graph of sentences and the edges between the sentences are connected based on a similarity measure. Using the concept of eigenvectors by **LexRank** [13] to the graph of sentences to estimate importance value for each sentence in the document. Using a topic-based approach like **LSA**, short for Latent Semantic Analysis [14], that tries to find sentences in the document by applying Singular Value Decomposition over document matrix D of size m x n where m is the number of sentences and n is the number of terms. Using a greedy search approximation approach like **SumBasic** [15] that uses a frequency-based sentence to set the weights of word probabilities to minimize redundancy. For each approach there is a literature review and the following sections below will describe the related work on each approach.

2.2 Statistical-based Approaches

1. Opinion Mining from online hotel reviews

(Y-H. Hu et al. 2017) proposes a multi-text summarization technique for identifying the top-k most informative sentences of hotel reviews [62]. The research process proposed was divided into five principal steps: hotel review collection, review preprocessing, sentence importance calculation, sentence similarity calculation, and top-k sentence recommendations. Their algorithm can be summarized into the following:

1. Data scraping: collecting hotel reviews from TripAdvisor.com
2. Data preprocessing: applying part-of-speech tagging, stop-words elimination, and POS filtering.

3. Sentence importance calculation: calculating a score for a sentence of a review based on the following features: 1) the representativeness of a review author, 2) helpfulness of a review, 3) review time, 4) the content of sentences in a review.
4. Sentence similarity calculation: the authors first identified that they want to calculate the content similarity which is the distance of each noun-noun pair between two sentences, calculated using Normalized Google Distance (NGD) [68].
5. Selection of Top-K sentences: this step applies the k-medoids algorithm algorithm to partition sentences into p clusters where p > selected k. each sentences is represented as a vector and the algorithm begins randomly selecting p sentences as an initial set of medoids and the cosine similarity between each sentence and each medoid is calculated until all medoids become stable. The importance score of a cluster can then be calculated by summing the importance score of each sentence and finally the top-k sentences can be selected by the top-k clusters exhibiting the highest importance scores.

The authors concluded that this was a study to focus on developing an extractive summarization technique which did not consider the context of the sentences, but introduces areas for improvement based on their proposed algorithm.

2. Extractive Text Summarization Using Sentence Ranking [69]

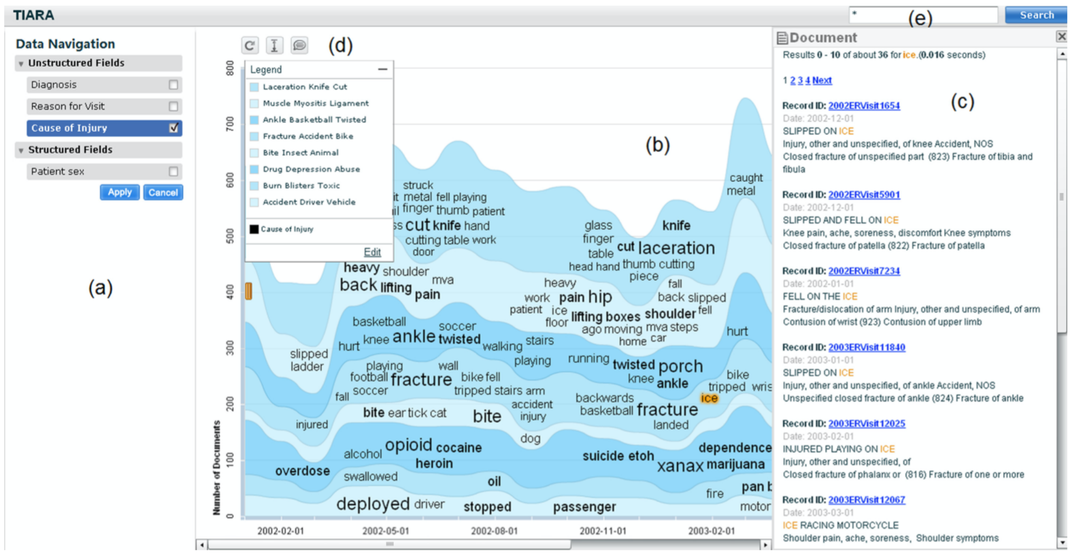
(J. Madhuri, G. Kuma, 2019) proposed a novel statistical method to perform extractive text summarization based on sentence ranking and produce a summary in an audio format. The steps of the proposed approach are the following:

1. Read the document and tokenize the input sentences.
2. Remove the stop words from the tokenized words.
3. Apply part-of-speech tagging to each key word in the input.
4. Calculate the frequency of each keyword.
5. Calculating weighted frequency of the word by dividing the frequency of the keywords by maximum frequency of the keywords.
6. Calculate the sum of weighted frequencies.
7. Generate a summary which will extract the high weighted frequency sentences and the extracted sentences into an audio format.

Their contribution was fairly straightforward which produced improved ROUGE scores when compared with the traditional approach [69].

3. TIARA: Interactive, Topic-Based Visual Text Summarization and Analysis [63]

(S. Liu et al., 2012) proposed an enhanced LDA-based topic analysis technique that automatically derives a set of topics to summarize a collection of documents and their content evolution over time. The authors studied different statistical approaches and wanted to present a visualized approach on information rather than simple analysis results such as the TFIDF algorithm. Their summarization approach is proposed as an interactive visual text analysis tool called TIARA (Text Insight Via Automated, Responsive Analysis) which is based on an enhanced topic modeling engine that summarizes the documents into a set of topics, each of which is represented by a set of keywords. The figure below shows the user interface of TIARA with a visual summary of the patients records from the National Hospital Ambulatory Medical Care Survey dataset. The authors concluded that their work contribution has presented a visual text summarization and an enhanced topic modeling technique which can produce time-sensitive and more meaningful text summaries.


TIARA
Data Navigation
Unstructured Fields
Diagnosis
Reason for Visit
Cause of Injury
Structured Fields
Patient sex
Apply
Cancel
(a)
(b)
(c)
(d)
(e)
Search
Legend
Laceration Knife Cut
Muscle Myositis Ligament
Ankle Basketball Twisted
Fracture Accident Bike
Bite Insect Animal
Drug Depression Abuse
Burn Blisters Toxic
Accident Driver Vehicle
Cause of Injury
Edit
Number of Documents
0
100
200
300
400
500
600
700
800
2002-02-01
2002-05-01
2002-08-01
2002-11-01
2003-02-01
struck metal fell playing finger thumb patient cut knife hand cutting table work door
glass finger table cut laceration knife head hand thumb cutting piece caught metal
heavy shoulder back lifting pain mva
heavy work pain hip patient ice floor lifting boxes shoulder fell back slipped fall ago moving mva steps home car
basketball soccer hurt knee ankle twisted walking stairs slipped ladder
playing wall football fracture bike fell soccer tripped stairs arm fall
running twisted porch knee ankle bike tripped wris ice hurt
backwards basketball fracture landed accident injury
injured bite ear tick cat bite dog
alcohol opioid cocaine heroin overdose swallowed oil
suicide etoh dependence xanax marijuana
deployed driver stopped passenger fire pan b motor

Document

Results **0 - 10** of about **36** for ice.(**0.016** seconds)

1 2 3 4 Next

Record ID: 2002ERVisit1654
Date: 2002-12-01
SLIPPED ON ICE
Injury, other and unspecified, of knee Accident, NOS
Closed fracture of unspecified part (823) Fracture of tibia and fibula

Record ID: 2002ERVisit5901
Date: 2002-12-01
SLIPPED AND FELL ON ICE
Knee pain, ache, soreness, discomfort Knee symptoms
Closed fracture of patella (822) Fracture of patella

Record ID: 2002ERVisit7234
Date: 2002-01-01
FELL ON THE ICE
Fracture/dislocation of arm Injury, other and unspecified, of arm
Contusion of wrist (923) Contusion of upper limb

Record ID: 2003ERVisit11840
Date: 2003-01-01
SLIPPED ON ICE
Injury, other and unspecified, of ankle Accident, NOS
Unspecified closed fracture of ankle (824) Fracture of ankle

Record ID: 2003ERVisit12025
Date: 2003-02-01
INJURED PLAYING ON ICE
Injury, other and unspecified, of
Closed fracture of phalanx or (816) Fracture of one or more

Record ID: 2003ERVisit12067
Date: 2003-03-01
ICE RACING MOTORCYCLE
Shoulder pain, ache, soreness, Shoulder symptoms

Figure 2: UI of the proposed tool: TIARA

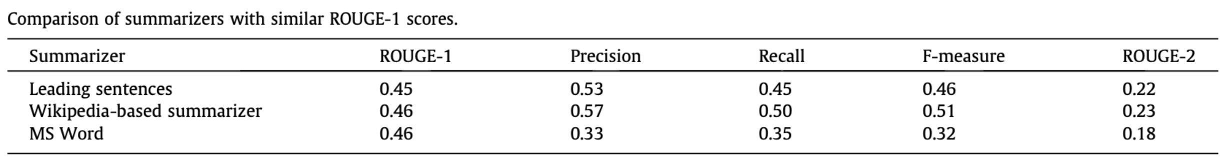
2.3 Graph-based Approaches

1. Wikipedia-based multi-document summarization [64]

(Y. Sankarasubramaniam et al. 2014) proposed an automatic summarization approach by first constructing a bipartite sentence-concept graph, and then rank the input sentences using interactive updates on their graph. The important feature that the authors mentioned in their contribution is that the proposed algorithm enables real-time incremental summarization in which users can first view an initial summary and then request additional content if interested. The proposed algorithm was trained on DUC 2002 dataset and has been evaluated using ROUGE scores and the authors concluded that the Wikipedia Summarizer was effective on both single-document and multi-document dataset and with the use of incremental summarization, the news articles are more easily read by the users. Their algorithm can be summarized into the following steps:

1. Given an input document D, the sentences are first parsed and mapped to Wikipedia Concepts in order to build a bipartite graph where one set of nodes represents the document's sentences and the other set of nodes represents the Wiki concepts. An edge between a sentence node and a concept node indicates a mapping between the corresponding document sentence and Wiki concepts.
2. Apply the proposed ranking algorithm which allows incremental summary generation.
3. Apply the summary extraction from the sentence-concept bipartite graph while incorporating the concept significance in the graph.

The figure below shows the results of the Wikipedia Summarizer with the baseline on the DUC 2002 dataset.



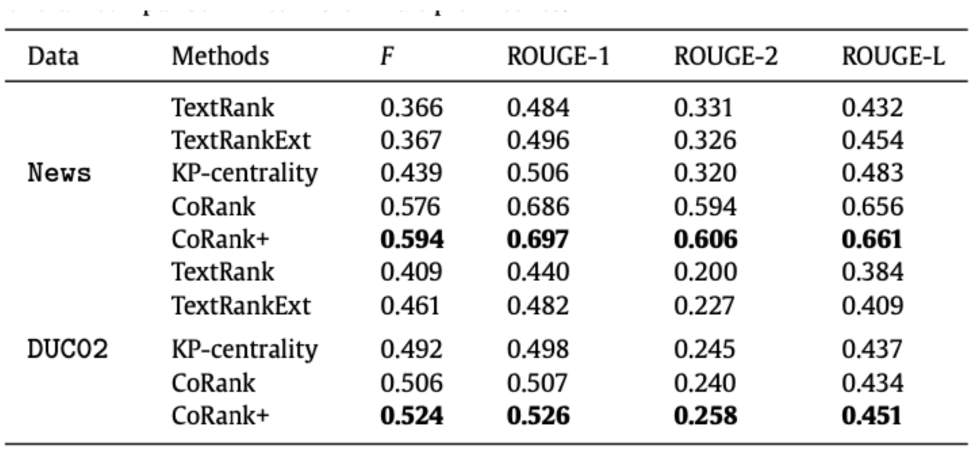
Comparison of summarizers with similar ROUGE-1 scores.

Summarizer	ROUGE-1	Precision	Recall	F-measure	ROUGE-2
Leading sentences	0.45	0.53	0.45	0.46	0.22
Wikipedia-based summarizer	0.46	0.57	0.50	0.51	0.23
MS Word	0.46	0.33	0.35	0.32	0.18

Table 1: ROUGE scores of the Wikipedia-based summarizer

2. CoRank: graph-based unsupervised ranking model [65]

(C. Fang, et al. 2017) proposed another graph-based approach for extractive summarization based on word-sentence co-ranking named CoRank. CoRank combines the word-sentence relationship with the graph-based unsupervised ranking model. The proposed approach incorporates the word-sentence relationship into the graph-based ranking model, such that the mutual influence is able to convey the intrinsic status of words and sentences more accurately. The authors then proposed to use the redundancy elimination technique for better selecting sentences as a summary, which can further improve the quality of generated summaries [65]. This approach was trained on two datasets called *News* containing 10 Chinese documents and *DUC2002* containing 567 English documents and the figure below shows the ROUGE scores of the author's proposed approach with respect to the baseline.



Data	Methods	*F*	ROUGE-1	ROUGE-2	ROUGE-L
News	TextRank	0.366	0.484	0.331	0.432
	TextRankExt	0.367	0.496	0.326	0.454
	KP-centrality	0.439	0.506	0.320	0.483
	CoRank	0.576	0.686	0.594	0.656
	CoRank+	**0.594**	**0.697**	**0.606**	**0.661**
DUC02	TextRank	0.409	0.440	0.200	0.384
	TextRankExt	0.461	0.482	0.227	0.409
	KP-centrality	0.492	0.498	0.245	0.437
	CoRank	0.506	0.507	0.240	0.434
	CoRank+	**0.524**	**0.526**	**0.258**	**0.451**

Table 2: ROUGE scores of the CoRank summarizer

3. EdgeSumm: Graph-based framework for automatic text summarization

(W.S. El-Kassas, et al. 2020) proposed a graph-based framework combining of four extractive algorithms (graph-based, statistical-based, semantic-based, and centrality-based) called EdgeSumm [67] which was reported to achieve performance scores higher than state-of-the-art systems in ROUGE-1 and ROUGE-2. The proposed framework consists of the following steps: 1) Pre-Processing, 2) Processing, and 3) Post-Processing. This proposed approach is meant to solve the summarization problem by introducing a generic framework for summarizing domain-independent documents, EdgeSumm is introduced as an unsupervised graph-based framework.

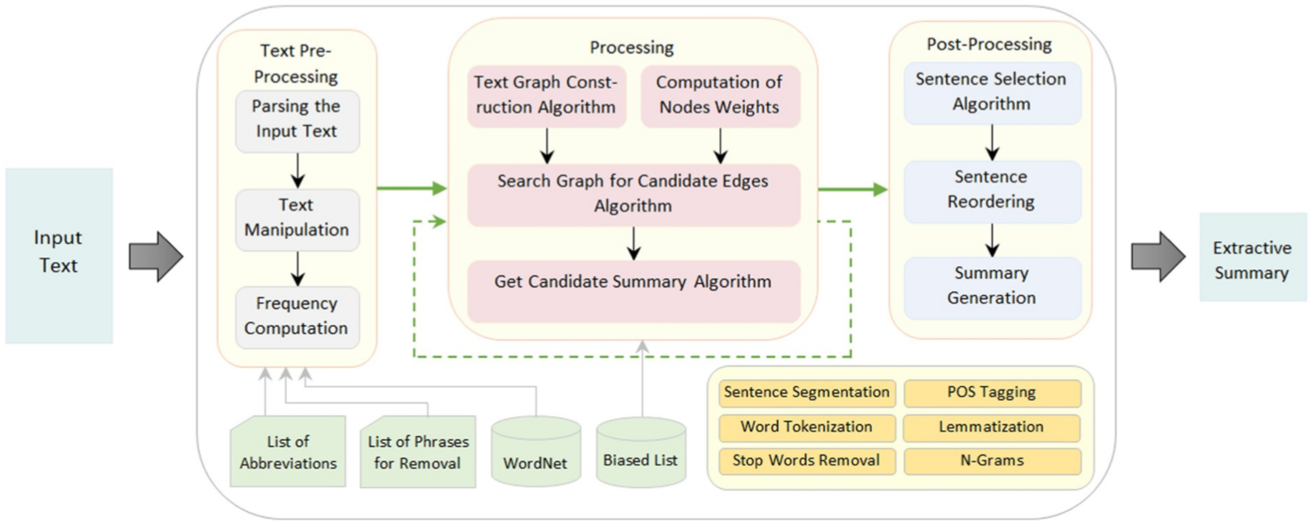


Figure 3: EdgeSumm Architecture

Summarization logic is applied in the second step which is the processing step as mentioned in the figure above. It first constructs a text graph model based on the output of the pre-processing step and then calculates the weight for each node in the graph based on the word frequency besides the occurrence of the word in the title and other important factors. Then a search graph algorithm is applied to try to find the phrases or edges that link the most frequent words or topics and the output of the algorithm is a list of selected edges for each sentence. Afterwards, a Candidate summary algorithm is applied which selects sentences to be included in the candidate summary based on which sentence has at least one edge selected in the candidate edges list from the search graph algorithm output. EdgeSumm has been applied on standard datasets DUC2001 and DUC2002 and it has outperformed the state-of-the-art text summarization systems for the ROUGE-1 and ROUGE-L metrics in the DUC2002 dataset.

4. Graph-based Extractive Summarizer using keywords or topic modeling [70]
(R. Belwal, et al. 2021) proposed a graph-based summarization technique which builds the graph where the weights of the edges are based on both the similarity between the sentences (nodes) and also considers the similarity between the sentences and the overall input document [70]. The generalized steps for a graph-based summarizer is the following:

- Representation of the sentences in the form of a graph where sentences are the nodes and the similarity measure to assign a weight to the edges.
- Applying graph-based algorithms to find the ranks of the nodes.

- Summary generation on the basis of rank.

However, what's different in the generalized steps here as mentioned in this proposed approach is that the similarity measure is dependent on both the sentences as well as the overall document. This approach was trained on CNN/DM dataset [17] and Opinosis dataset [8] and has produced ROUGE scores in the experiments section which demonstrated that the proposed algorithm achieves better performance than the existing graph-based methods in terms of ROUGE matrices.

2.4 Deep learning-based Approaches

1. SummaRuNNer - Deep Learning Extractive Summarizer

(Nallapati R, Zhai F, Zhou B 2017) proposed a deep learning model called SummaRuNNer [3]. It is a neural network approach that treats the problem as a sequence classification problem. It is a two-layer recurrent neural network (RNN) based sequence classifier where the bottom layer operates at word level within each sentence, while the top layer operates at sentence level. Every sentence is visited sequentially in the original document order and a binary decision is made while taking into account previous decisions made to give a verdict whether or not it should be included in the summary. This approach used a Gated Recurrent Units (GRU) as a building block for their network [16]. A GRU-RNN can be described as follows. The Weights of the network W's and biases b's are parameters of the GRU-RNN where h_j is the real-valued hidden-state timestep j and x_j is the input vector after word embedding. GRU-RNN has two gates, **u** called the update gate and **r** which is the reset gate and their equations are represented as follows.

Update gate at timestep j.

$$\mathbf{u}_j = \sigma(\mathbf{W}_{ux}\mathbf{x}_j + \mathbf{W}_{uh}\mathbf{h}_{j-1} + \mathbf{b}_u)$$

Reset gate at timestep j.

$$\mathbf{r}_j = \sigma(\mathbf{W}_{rx}\mathbf{x}_j + \mathbf{W}_{rh}\mathbf{h}_{j-1} + \mathbf{b}_r)$$

Forward Hidden layer at time step j.

$$\mathbf{h}'_j = \tanh(\mathbf{W}_{hx}\mathbf{x}_j + \mathbf{W}_{hh}(\mathbf{r}_j \odot \mathbf{h}_{j-1}) + \mathbf{b}_h)$$

Backward Hidden layer at time step j.

$$\mathbf{h}_j = (1 - \mathbf{u}_j) \odot \mathbf{h}'_j + \mathbf{u}_j \odot \mathbf{h}_{j-1}$$

The entire document can be represented as a nonlinear function of the average pooling of the concatenated hidden states of the bi-directional sentence-level RNN as follows:

$$\mathbf{d} = \tanh(W_d \frac{1}{N_d} \sum_{j=1}^{N^d} [\mathbf{h}_j^f, \mathbf{h}_j^b] + \mathbf{b}),$$

For classification, each sentence is revisited in a second pass with a logistic layer that makes a binary decision whether sentence$_i$ should belong to the summary.
Below is the final architecture of the SumaRuNNer where the bottom layer handles word level for each sentence, and the top layer handles the sentence level, and after document representation it gives a binary decision to each sentence whether it should be included or not. The binary classification assigns 1 for included and 0 for not included.

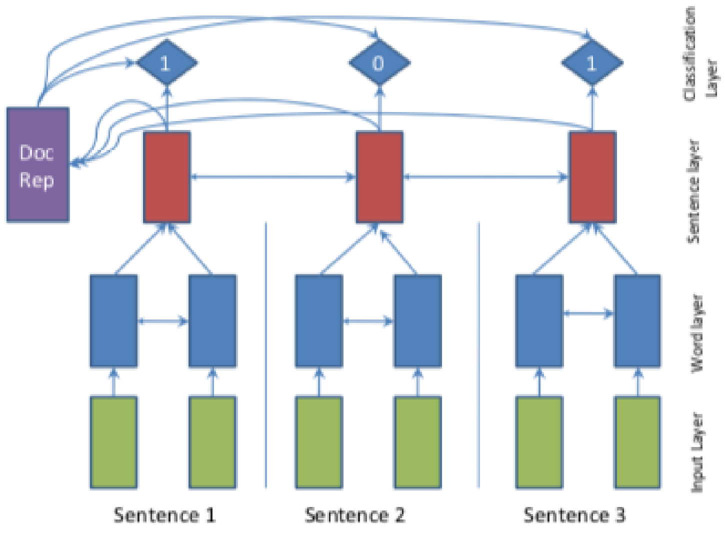


Figure 4: SummaRuNNer: A Recurrent Neural Network based Sequence Model for Extractive Summarization

2. Attentional Encoder-Decoder Extractive Summarizer
(Cheng and Lapata 2016) proposed a similar deep learning approach. They proposed an attentional encoder-decoder for extractive single-document summarization and applied to the CNN/DailyMail corpus [17]. This paper proposed the CNN and Daily Mail corpus and their work is the closest to the SummaRuNNer model. Their model is based on an encoder-decoder approach where the encoder learns the representation of sentences and documents while the decoder classifies each sentence based on the encoder's representation using an attention

mechanism. In this paper, they built multiple representations using neural networks. They first obtained representation vectors at the sentence level using a single-layer convolutional neural network (CNN) as they can be trained effectively without long-term dependencies in the model and successful for sentence-level classification. Afterwards, they built representations for documents using RNN that recursively composes sentences [17]. First step is called Convolutional Sentence Encoder, the second step is called Recurrent Document Encoder. Third step is Sentence Extractor which is another RNN that applies attention mechanisms to directly extract salient sentences after reading them. The labeling decision is made with both the encoded document and the previously labeled sentences in mind. After this sequence labeling task, the fourth step is Word Extractor. This task is responsible for generating the next word in the summary using a hierarchical attention architecture and computes the probability of the next work to be included in the summary [17].

First figure shows the first three steps, the CNN sentence encoder followed by the RNN document encoder, followed by the RNN sentence extractor with attention mechanism.

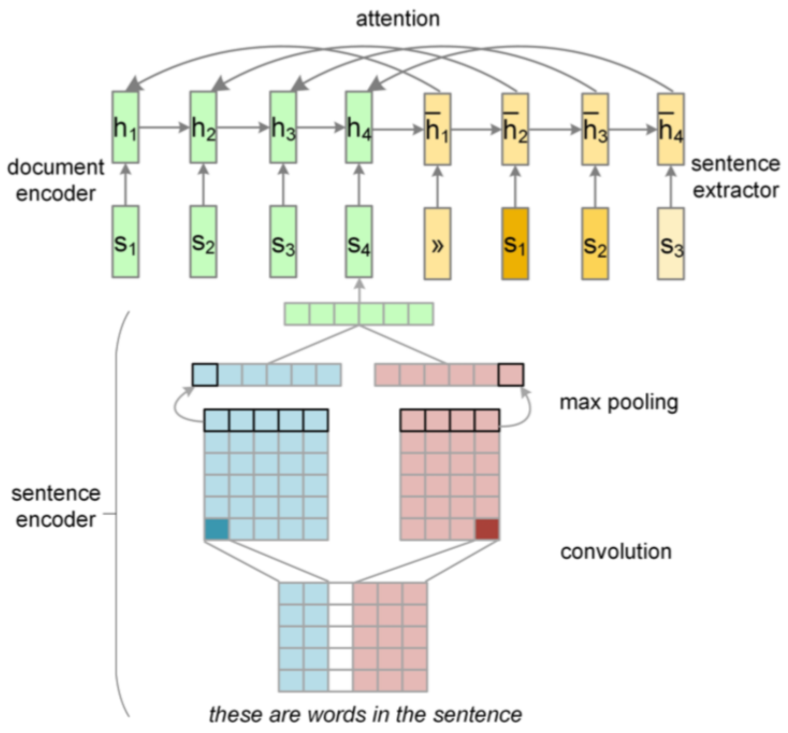
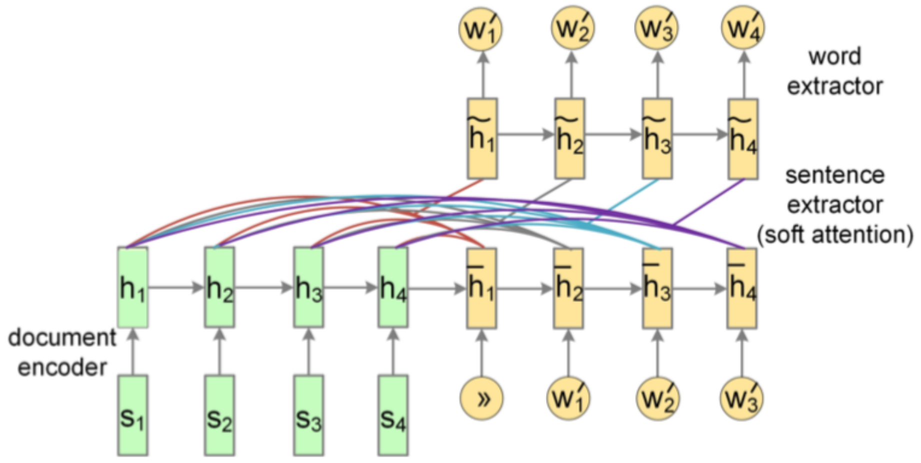


Figure 5: Attentional Encoder-Decoder Architecture, CNN + RNN

Second figure shows the labeling process that happens from sentence extractor phase to word extractor phase. Both figures together are the full architecture of this paper.



w'_1 w'_2 w'_3 w'_4
word extractor
$\tilde{h}_1$ $\tilde{h}_2$ $\tilde{h}_3$ $\tilde{h}_4$
sentence extractor (soft attention)
h_1 h_2 h_3 h_4 $\bar{h}_1$ $\bar{h}_2$ $\bar{h}_3$ $\bar{h}_4$
document encoder
s_1 s_2 s_3 s_4 » w'_1 w'_2 w'_3

Figure 6: Attentional Encoder-Decoder Architecture, Neural Summarization by Extracting Sentences and Words

3. SummCoder - Unsupervised Extractive Summarizer

(Joshi, Eduardo, Enrique, and Laura 2019) proposed SummCoder [18], an unsupervised framework for extractive text summarization based on deep auto-encoders.

In this approach, the extractive summarization problem was defined as a sentence sequence ranking or sentence selection problem in a document. The determination of which sentences should be included in the summary was based on three criteria:

i) Sentence Content Relevance Metric
ii) Sentence Novelty Metric
iii) Sentence Position Relevance Metric

This proposed framework ranks sentences based on these three metrics. Based on this paper's problem formulation, it describes the following: Consider a document D as a bag of textual units and each textual unit as a bag of sentences. Let a textual unit be denoted as $T \in D$, and a sentence as $S \in T \in D$. Therefore, a document can be considered as a bag of sentences. If a document has N sentences $D = (S_1, S_2, ..., S_N)$, each sentence is embedded to a vector which is encoder phase, and then it gets computed with the three metrics mentioned before and decoded afterwards. The following figure is their introduced SummCoder Framework [18].

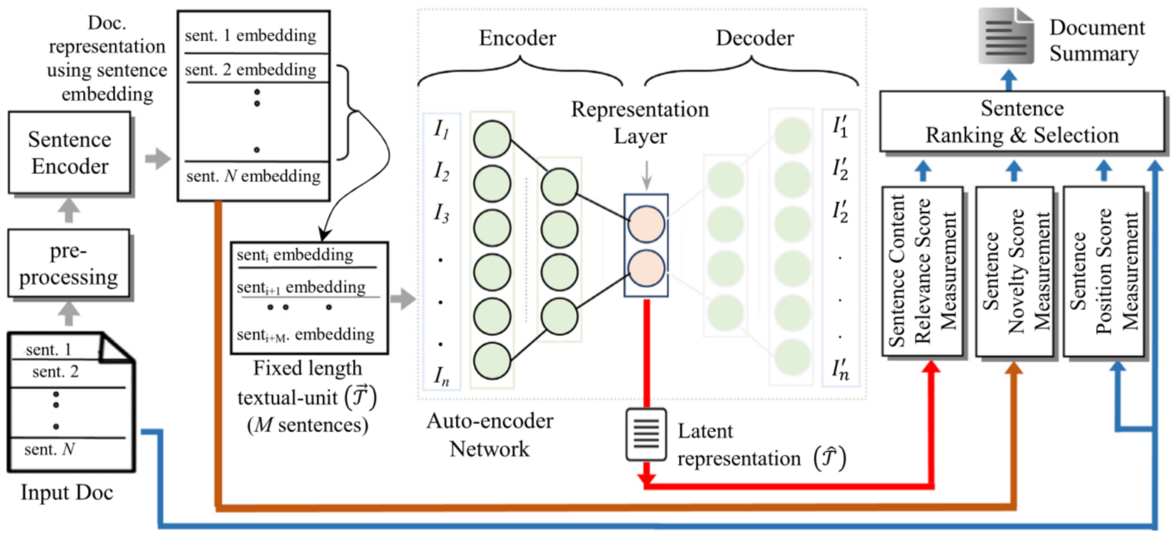

Doc. representation using sentence embedding
sent. 1 embedding
sent. 2 embedding
sent. N embedding
Sentence Encoder
pre-processing
sent. 1
sent. 2
sent. N
Input Doc
sent$_i$ embedding
sent$_{i+1}$ embedding
sent$_{i+M}$ embedding
Fixed length textual-unit ($\vec{\mathcal{T}}$) (M sentences)
Encoder
Decoder
Representation Layer
I_1 I_2 I_3 I_n
I'_1 I'_2 I'_2 I'_n
Auto-encoder Network
Latent representation ($\hat{\mathcal{T}}$)
Document Summary
Sentence Ranking & Selection
Sentence Content Relevance Score Measurement
Sentence Novelty Score Measurement
Sentence Position Score Measurement

Figure 7: SummCoder: An unsupervised framework for extractive text summarization based on deep auto-encoders

4. BanditSum - Extractive Summarization As Contextual Bandit Problem

(Yue Dong, et. al 2019) proposed a novel method for training neural networks to perform single-document extractive summarization without heuristically-generated extractive labels. They call their approach BanditSum as it treats extractive summarization as a contextual bandit (CB) problem, where the model receives a document to summarize (the context), and chooses a sequence of sentences to include in the summary (the action) [27].

The paper formulates extractive summarization as a contextual bandit where they trained an agent to solve using policy gradient reinforcement learning. A bandit is a decision-making formalization in which an agent repeatedly chooses one of several actions, and receives a reward based on this choice. For extractive summarization tasks, each document is labeled as a context and each ordered subset of a document's sentences is a different action. The agent is required to learn quickly from a series of actions which yield the highest reward. This approach has shown promising results and they have shown empirically that BanditSum performs significantly better than competing approaches when good summary sentences appear late in the source document.

5. NeuSUM - Document Encoder and Sentence Extractor As Extractive Summarizer

(Qingyu, et. al 2018) proposes a different neural network approach for extractive text summarization called NeuSUM [28]. In this paper they present a novel end-to-end neural network framework for extractive document summarization by jointly learning to score and select sentences. It first reads the document sentences with a hierarchical encoder to obtain the representation of sentences. Then it builds the output summary by extracting sentences one by one. Their proposed model consists of two parts, the document encoder and the sentence extractor. The document encoder has a hierarchical architecture, which suits the compositionality of documents. The sentence extractor is built with RNN architecture. They conducted experiments on the CNN/Daily Mail dataset and their proposed model at the time of publication has significantly outperformed state-of-the-art methods and achieved the best result on CNN/Daily Mail dataset. [28]

2.5 BERT-based Approaches

Self-Attention Networks:

In most BERT-derived networks there are typically 3 stages: the embedding, the encoder, and the classifier [25]. The embedding converts preprocessed words (represented as integer-valued tokens) into learned feature-vectors of the floating-point numbers; the encoder consists of a series of self-attention and other layers; and the classifier produces the network's final output. Since the embedding and the classifier account for ~ 1% of the runtime of a self-attention network, we focus on the encoder [45].

We now describe the encoder that is used in BERT-base. The encoder consists of a stack of blocks. Each block consists of a self-attention module followed by three position-wise fully-connected layers, known as feed-forward network (FFN) layers. Each self-attention module contains three separate position-wise fully-connected (PFC) layers, which are used to generate the query (Q), key (K), and value (V) activation vectors for each position in the feature embedding. Each of these PFC layers in self-attention applies the same operation to each position in the feature embedding independently. While neural networks traditionally multiply weights by activations, a distinguishing factor of attention neural networks is that they multiply activations by other activations, which enables dynamic weighting of tensor elements to adjust based on the input data. Further, attention networks allow modeling of arbitrary dependencies irregardless of their distance in the input or output. **(Vaswani et al.)** proposed the self-attention module [46] which is also used in different BERT models such as GPT, BERT, RoBERTa, ELECTRA, and others [47][25][49][48]. It multiplies the Q, K, V activations vectors together using the equation

$$Softmax(\frac{QK^T}{\sqrt{d_k}})$$

where d_k is the number of channels in one attention head.

1. BERTSum - BERT-based Extractive Summarization Model

(Yang Liu 2019) offers in his paper a variant of extractive summarization approach using contextualized-word embeddings BERT. In his paper, he fine tuned a bert based model to perform extractive text summarization called BERTSum [31]. BERT is a pre-trained Transformer model that has achieved ground-breaking performance on multiple NLP tasks, and the model has outperformed several approaches in this literature review. In his methodology, he uses BERT to output representation for each sentence since BERT is trained as a masked-language model and then he modified the input sequence and embeddings of BERT to make it possible for extracting summaries. The figure below shows the complete architecture of BERTSum where the model inserts a [CLS] token before each sentence and a [SEP] token after each sentence, and then adds three embedding layers; Token Embeddings, Interval Segment Embeddings, and Position Embeddings as input to the BERT layer.

The output of the BERT layer is sentence vectors from BERT, which steps into several summarization-specific layers stacked on top of the BERT outputs, to capture document-level features for extracting summaries. The author calculates the final predicted score Y for each sentence Si and computes the binary classification entropy as the loss function to label whether this sentence should be selected or not. [31]

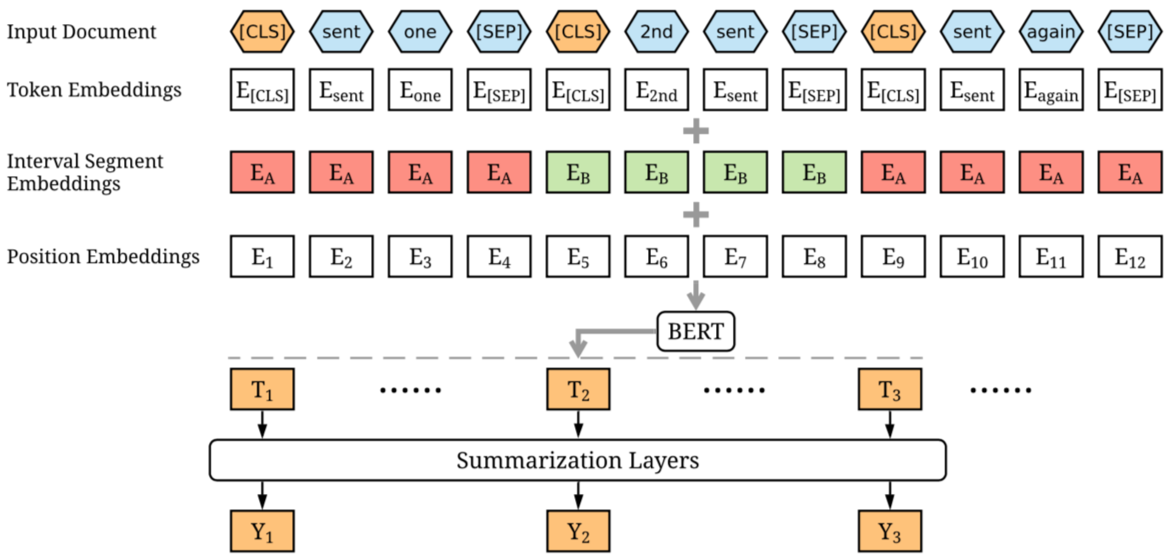


Figure 8: BERTSum Architecture

2. Automatic Text Summarization of COVID-19 Medical Research Articles using BERT and GPT-2

With the emergence of the COVID-19 pandemic, **(Tan, Kieuvongngam and Niu 2020)** published a paper on solving the text summarization problem for the COVID-19 Open Research Dataset dataset [35] which has released a corpus of scholarly articles. The authors have taken advantage of the recent advances in NLP models such as BERT and OpenAI GPT-2 to perform text summarization of both abstractive and extractive approaches [59]. The authors used DistilBERT model for unsupervised extractive summarization implemented in the Hugging Face transformer [55]. For the abstractive summarization, the authors used a pre-trained distil version of GPT-2 model called DistilGPT2 also implemented in the Hugging Face transformer [55]. After training their models and evaluating them using the ROUGE metric, the authors concluded that there is a room for improvement by training a complex BERT models such as the bigger GPT-2 version with more intensive computation resources to rapidly improve abstractive summarization to fight the pandemic and expand the literature on COVID-19 dataset.

3. Combination of abstractive and extractive approaches for summarization of long scientific texts [60]

(Tretyak Vladislav and Stepanov Denis 2020) presented a comparison analysis of different architectures for extractive and abstractive summarization approaches which inspired the idea of developing the second research question for this thesis regarding choosing BERT variant for summarization. The authors also proposed a method for generating summaries of long texts by using both an extractive model and an abstractive model together as seen in the figure below.

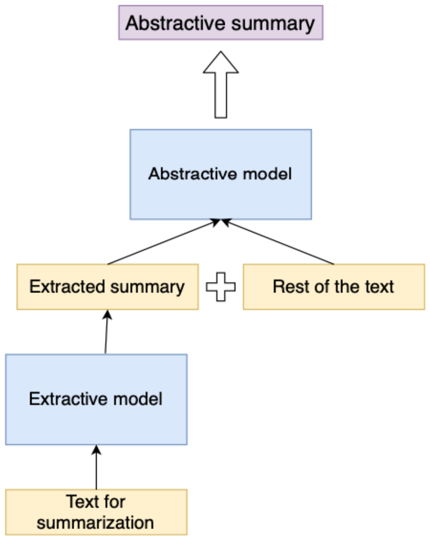


Figure 9: Proposed Architecture for Abstractive Summarization of long documents

After performing the experiments on extractive summaries, the authors concluded that the introduction and the conclusion of the documents showed the best rouge score and also proposed the new architecture which could improve the quality of abstractive summaries. Their experiments were evaluated using ROUGE-1, ROUGE-2, and ROUGE-L and they used the arxiv dataset in their research.

4. DiscoBERT - Discourse-aware Neural Extractive Summarization Model

Since BERT has been adopted for document encoding in state-of-the-art text summarization models. **(Jiacheng Xu, et. al 2020)** published a paper that presents a discourse-aware neural summarization model or as the authors called it - DiscoBert [29]. DiscoBert extracts sub-sentential discourse units (instead of sentences) as candidates for extractive selection on a finer granularity. To capture the long-range dependencies among discourse units, structural discourse graphs are constructed based on RST trees and coreference mentions, encoded with Graph Convolutional Networks. Experiments show that the proposed model outperforms state-of-the-art methods by a significant margin on popular summarization benchmarks compared to other BERT-base models.

5. MatchSum - Extractive Summarization As Text Matching Problem

(Ming Zhong, et. al 2020) presented a paper that tackles extractive text summarization in a different approach as a Text Matching problem, the authors called it MatchSum [30]. This paper creates a paradigm shift with regard to the way neural extractive summarization systems. Instead of following the commonly used framework of extracting sentences individually and modeling the relationship between sentences, they formulated the extractive summarization task as a semantic text matching problem in which a source document and candidate summaries will be (extracted from the original text) matched in a semantic space.

Experiments have been done on five datasets which show the effectiveness of the matching framework and they believe the power of this matching-based summarization framework has not been fully exploited.

The figure below demonstrates the authors' methodology with the MatchSum framework. They match the contextual representations of the document with the gold summary and candidate summaries and they claim intuitively that better candidate summaries should be semantically closer to the document, while the gold summary should be the closest. [30]

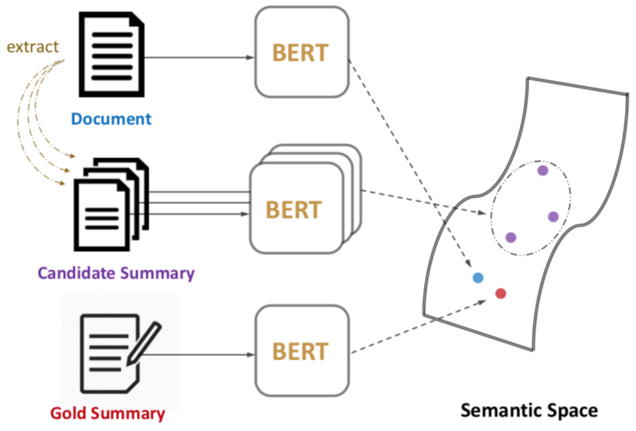


Figure 10: Overview of MatchSUM framework

The following table below shows the comparison of different extractive text summarization approaches on the CNN/DM dataset using ROUGE as the prime evaluation metric. This table shows a comparison of the approaches mentioned in this literature review with ROUGE scores published by the authors [17][27][28][29][30][31]. The related works are sorted in descending order but the computation time needed to train each approach is not recorded in this table.

Model	ROUGE-1	ROUGE-2	ROUGE-L
MatchSum (Zhong et al., 2020)	44.41	20.86	40.55
DiscoBERT (Xu et al. 2019)	43.77	20.85	40.67
BertSum (Liu and Lapata 2019)	43.85	20.34	39.90
NeuSUM (Zhou et al., 2018)	41.59	19.01	37.98
BanditSum (Dong et al., 2018)	41.5	18.7	37.6
SummaRuNNer (Nallapati et at., 2017)	39.6	16.2	35.3
Lead-3 baseline (Nallapati et al., 2017)	39.2	15.7	35.5

Table 3: Comparison of different extractive summarization approaches

2.6 Word Embedding

Textual data has to be represented in a way that the machine can understand. Word embedding helps this transformation by representing each word as a one-hot vector of a certain size. That one-hot vector has only one entry set to 1 which represents the position or index of that word and all the other entries are set to 0. This is how a word can be embedded to a one-hot vector which can later be used as trainable data. It improves the performance of the text summarization process and it is applicable to compute two-word similarity with their vector representation.

2.6.1 Non-Contextualized Embeddings

Word2Vec

Word2Vec is a continuous bag-of-words or in short, CBOW, architecture for creating word embeddings that uses *n* future words as well as *n* past words to create a word embedding. The objective function for CBOW is:

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} logp(wt \mid wt - n, ..., wt - 1, wt + 1, ..., wt + n)$$

(Mikolov et al., 2013) proposed word2vec as a method for efficient estimation of word representation in vector space [11].

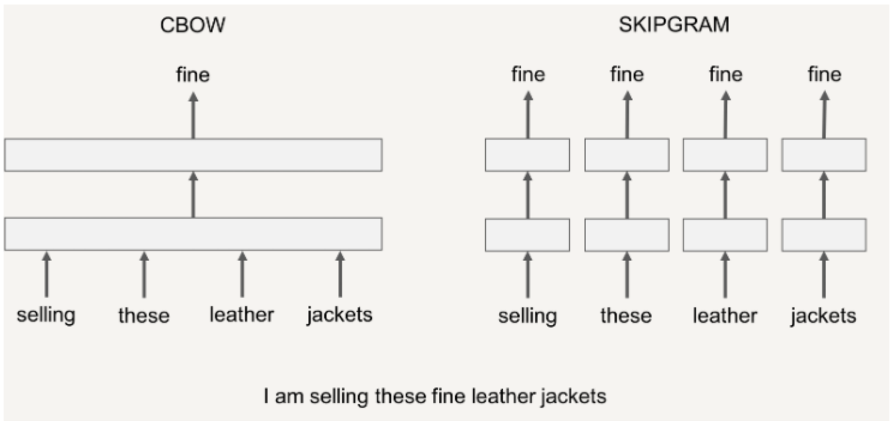


Figure 11: Overview of Word2Vec

GloVe

GloVe Embeddings are a type of word embedding that encodes the co-occurrence probability ratio between two words as vector differences. GloVe uses a weighted least squares objective J that minimizes the difference between the dot product of the vectors of two words and the logarithm of their number of co-occurrences:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \hat{w}_j + b_i + b_j - logX_{ij})^2$$

where w_i and b_i are the word vector and bias respectively of word i, ŵ , j and b_j are the context word vector and bias respectively of word k, X_{ij} is the number of times word i occurs in the context of word j, and f is a weighting function that assigns lower weights to rare and frequency co-occurrences [23].

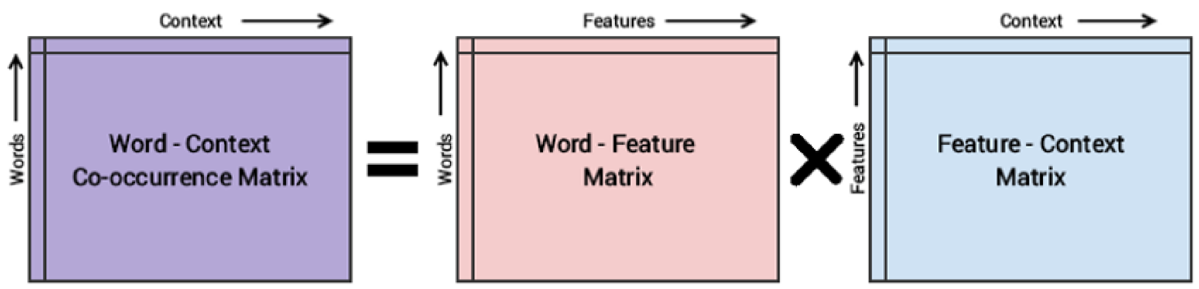


Figure 12: Overview of GloVe

2.6.2 Contextualized Embeddings

ELMo

ELMo is a deep contextualized word representation that models both (1) complex characteristics of word use syntactically and semantically, and (2) how these uses vary across linguistic contexts. These word vectors are learned functions of the internal states of a deep bidirectional language model (biLM), which is pre-trained on a large text corpus [24]. They can be easily added to existing models and significantly improve the SoTA across a broad range of challenging NLP problems. (Peters et al., 2018) summarized the major advantages of ELMo embeddings in the following:

- Contextual: the representation of each word depends on the entire context in which it is used.
- Deep: the word representations combine all layers of a deep pre-trained neural network
- Character based: ELMo representations are purely character based, allowing the network to use morphological clues to form robust representations for out-of-vocabulary tokens unseen in training.

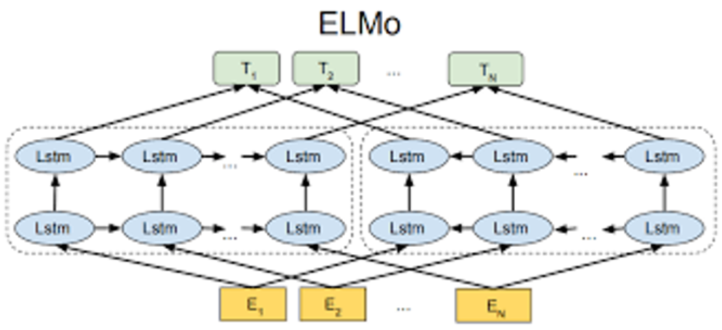


Figure 13: Overview of ELMo

BERT

BERT is one of the major transfer learning models that made a breakthrough in the year of 2018 for NLP tasks along with Allen AI's ELMo. BERT by Google is a bidirectional encoder representation from transformers. BERT can be used to extract high quality language features from textual data, i.e., more representative word embeddings, or it can be fine-tuned on a specific task such as classification, question-answering, or NER. BERT has been proved to be useful in SoTA systems and it offers an advantage over other language models like word2vec. BERT produces contextualized word embeddings such that it captures other forms of information around it in a given sentence that result in more accurate feature representations, which in turn results in better model performance [25].

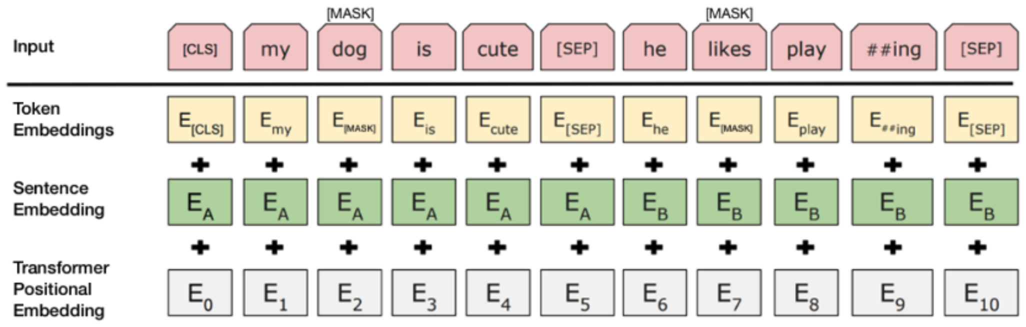


Figure 14: Overview of BERT

2.7 ROUGE Evaluation Algorithm

Generating a summary does not have absolute correctness. Each summary made by a human reader differs from another based on what he/she perceives as important information to include or what are the unique sentences that have to be included. In text summarization, an evaluation metric to use is Recall-Oriented Understudy for Gisting Evaluation, or in short, ROUGE recall measure [19]. Rouge-N is an N-gram recall measure between the golden summary and the generated summary. It is the ratio of count of N-gram phrases which occur in both the golden and the generated summary. In other words, the ROUGE-N metric measures the overlap between the golden summary (reference summary) and generated summary.

Variants of ROUGE metric like:

- Rouge-N: measures unigram, bigram, trigram, or n-gram in general.
    - Rouge-1 (unigram)
    - Rouge-2 (bigram)

- ○ Rouge-3 (trigram)

The general equation for Rouge-N is as follows:

$$\frac{\sum_{S\in(\text{Reference Summary})}\sum_{\text{N}-\text{gram}\in S}\text{Count}_{\text{match}}(\text{N}-\text{gram})}{\sum_{S\in(\text{Reference Summary})}\sum_{\text{N}-\text{gram}\in S}\text{Count}(\text{N}-\text{gram})}$$

- ● ROUGE-L: measures longest common subsequence (LCS) of words.

Rouge-L assesses the fluency of the summary through measuring the longest matching sequence of words using LCS. In LCS, the matches are not consecutive, but in-sequence reflects sentence level word order. ROUGE-L is computed for golden summary X of length m and predicted summary Y of length n as follows:

$$\frac{\text{LCS}(X,Y)}{m}$$

Evaluation of a summary can also be evaluated by computing the value of precision, recall, and F-score. Precision is used to determine the level of accuracy between documents requested and the answers given by the system Precision calculation using the formula:

$$Precision\ (P)\ = \frac{S_{ref} \cap S_{cand}}{S_{cand}}$$

The recall is used for documents that are recalled and are relevant to the query entered by the user on the system. The recall score is calculated using the formula:

$$Recall\ (R)\ = \frac{S_{ref} \cap S_{cand}}{S_{ref}}$$

F-scores are obtained by combining precision and recall. How to calculate the F-score is to calculate the average precision and recall. The formula below:

$$F\ -\ Measure\ = \frac{2\ (P)\ (R)}{P + R}$$

Chapter 3: Methodology

This chapter presents and classifies each experiment under the two research questions mentioned section 1.4. It presents the objectives of each of the experiments, along with the tools needed and the steps for implementation.

3.1 Preliminary Experiment - Studying the impact of different models on CNN/DM dataset

The objective of this experiment is to explore BertSum, a variant of BERT for extractive summarization [31]. The source code is publicly available in Python 3.6 with required libraries to install; pytorch, pytorch_pretrained_bert, tensorboardX, multiprocess, and pyrouge. This experiment will run on the available GPU resource by Google Colab notebooks.

The task of extractive summarization is a binary classification problem at the sentence level. Given a document D with N sentences, we want to assign each sentence with a label Y ε {0,1} to classify whether this sentence should be included in the final summary or not.

BERTSum is an extension of the original BERT model focused on the task of summarization, the difference between the two architectures is that the BERTSum model adds symbols to the input to represent the start and the end of a sentence so that the model can learn sentence representations. BertSum also embeds pairs of sentences to learn adjacency patterns between each input sentence in the segment embedding layer. The objective of this model is to assign every sentence in the document a score representing the relevance of the sentence to the overall document to classify which sentence is included in the summary or not. The figure below shows the differences mentioned between the original BERT architecture and the BERTSum architecture [31].

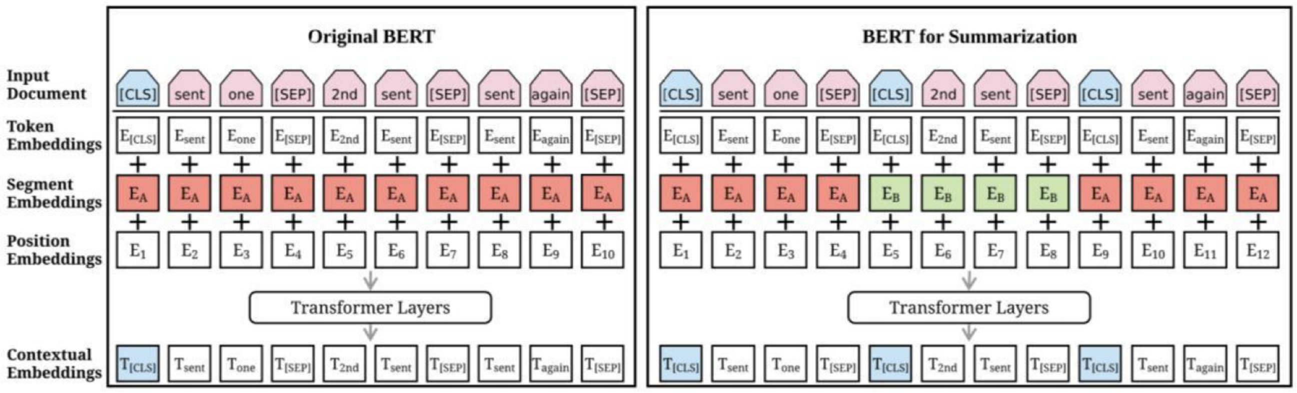


Figure 15: Comparison between BERT architecture and BERTSumm architecture

The figures below show the BERTSum algorithm in fine tuning the layers of the original BERT for the task of summarization. The algorithm includes placing the positional embeddings to indicate the location of the sentence in the document, also known as document contextualization.

$$\begin{aligned}\tilde{h}^l &= \mathrm{LN}(h^{l-1} + \mathrm{MHAtt}(h^{l-1})) \\ h^l &= \mathrm{LN}(\tilde{h}^l + \mathrm{FFN}(\tilde{h}^l))\end{aligned}$$

*Figure 16: Fine Tuning Equation for Layer **l** in Transformer Layers*

These equations above enable us to utilize the classifier to calculate the score ŷ for sentence i. By ranking the sentence scores, we will be able to identify the highest ŷ sentences scores which will formulate the output summary of the BERTSum model. The figure below shows the scoring formula of the BERT architecture.

$$\hat{y}_i = \sigma(W_o h_i^L + b_o)$$

Figure 17: Sentence Scoring Equation in BERT.

3.2 Choosing best variant of BERT model to produce better performance

This section presents a detailed objective of each experiment in answering the second research question mentioned in section 1.4. It comprises three experiments where each experiment consists of training a variant of the bert model and evaluating its performance using the ROUGE metrics. The experiments have been conducted and evaluated using the Google Colab Notebook.

3.2.1 Experiment 2 - Fine-Tuning DistilBERT on CNN/DM dataset

The objective of this experiment is to explore DistilBERT. DistilBERT is a compressed variant of BERT used to reduce the computation cost of training an extractive summarizer and produce a summary while maintaining the model's performance to be deployed on low-resources devices. DistilBERT being smaller than BERT-base by ~40% [33], the objective here is to modify BertSum to replace the BERT base encoder with DistilBERT and re-train the model and record the ROUGE scores. This experiment will run on the available GPU resource by Google Colab notebooks. DistilBERT aims to leverage knowledge distillation during the pre-training phase and show that it is possible to reduce the size of a BERT model by 40% while retaining 97% of its language understanding capabilities and being 60% faster. Knowledge distillation was first introduced by *Bucila et al, 2006* [36][37] which is a compression technique in which a compact model called, the student, is trained to reproduce the behaviour of a larger model, i.e., the teacher, or an ensemble of models.

DistilBERT was trained using a triple loss which combines language modeling, distillation, and cosine-distance losses. The student is trained with a distillation loss over the soft target probabilities of the teacher:

$$Lce = \sum_{i} ti * log(si)$$

where **ti** is a probability estimated by the teacher with respect to the student. The final training objective is a linear combination of the distillation loss **Lce** with the supervised training loss, which is the masked language modeling loss **Lmlm** [25] as well as adding the cosine embedding loss **Lcos** which helps in aligning the direction of both the student and the teacher's hidden states vectors.

The architecture of DistilBERT is introduced as the "Student" architecture, which has the same general architecture as BERT, except for the following changes:

1. The token-type embeddings and the pooler are moved.
2. The number of layers is reduced by a factor of 2.

The authors emphasized on reducing the number of layers as the major change in the BERT architecture as well as introducing a triple loss which combines language modeling, distillation and cosine-distance losses.

3.2.2 Experiment 3 - Fine-Tuning SqueezeBERT on CNN/DM dataset

The objective of this experiment is to explore SqueezeBERT. The SqueezeBERT model was proposed in paper [32] to reduce the computationally expensive training of complex architectures such as BERT. SqueezeBERT architecture uses grouped convolutions instead of fully-connected layers for the Q, K, V, and FFN layers. The paper discusses an observation that methods such as grouped convolutions have yielded significant speedups for computer vision networks and they demonstrate a replacement of several operations in self-attention layers with grouped convolutions and proposed this novel network architecture called SqueezeBERT which reportedly runs 4.3x faster than BERT-base. In this experiment, I modified BERTSum to replace the BERT base encoder with SqueezeBERT and trained the model and recorded the ROUGE scores. This experiment will run on the available GPU resource by Google Colab notebooks.

To further explain SqueezeBERT architecture, it started with a pretrained neural network called SqueezeNet [38] which was introduced by the computer vision community for optimizing neural networks on mobile devices. The authors of this research were encouraged by the progress of leveraging ideas that are popular in the CV literature to accelerate NLP, and applied two ideas in SqueezeBERT.

1. Convolutions: computer vision neural networks have relied heavily on convolutional layers since the 1980s [39][40]. Convolutions are quite flexible and well-optimized in software, and they can implement things as simple as a 1D fully-connected layer, or as complex as a 3D dilated layer that performs upsampling or downsampling.
2. Grouped Convolutions: a popular technique in model mobile-optimized neural networks. **(Krizhevsky et al., 2012)** proposed this technique in the winning submission to the ImageNet Image classification challenge [41] and it is extensively used in efficient CV networks such as MobileNet[42], ShuffleNet[43], and EfficientNet[44]. Due to the successful usage of Grouped Convolutions in different architecture in CV literature, the authors of SqueezeBERT have applied grouped convolution to NLP as part of their contribution.

SqueezeBERT architecture is still similar to the BERT base architecture, but with the position wise fully connected layers, or PFC layers, implemented as convolutions, and grouped convolutions for many of the layers. Each block in the BERT base encoder has a self-attention module with 3 PFC layers, plus 3 more PFC layers called feed-forward network layers (FFN1, FFN2, and FFN3). The FFN layers have the following dimensions: FFN1 has Cin=Cout=786, FFN2 has Cin=768 and Cout=3072, and FFN3 has Cin=3072 and Cout=768. In all PFC layers of the self-attention modules, and in the FFN2 and FFN3 layers, they used grouped convolutions with G = 4. To allow for mixing across channels of different groups, we use G=1 in the less-expensive FFN1 layers. Note that in BERT base, FFN2 and FFN3 each have 4 times more arithmetic operation than FFN1. However, when we use G = 4 in FFN2 and FFN3, now all FFN layers have the same number of arithmetic operations. Finally, the embedding size (768), the number of blocks in the encoder (12), the number of heads per self-attention module (12), the tokenizer (WordPiece [50]), and other aspects of SqueezeBERT are adopted from BERT base architecture. The authors have used a combination of Wikipedia and BooksCorpus [51] for the pre-training phase of SqueezeBERT, and have set aside 3% of the combined dataset as a test set. In classification tasks, SqueezeBERT was trained using the soft cross entropy loss with respect to a weighted sum of the teacher's logits and a one-hot encoding of the ground-truth.

3.2.3 Experiment 4 - Fine-Tuning MobileBERT on CNN/DM dataset

The objective of this experiment is to explore MobileBERT [34]. MobileBERT is a compressed and accelerated version of the BERT model. The authors describe it as a Task-agnostic approach, which means it can generically be applied to various downstream NLP tasks via simple fine-tuning. For our use case, we will fine-tune MobileBERT for our summarization experiment. MobileBERT was introduced before SqueezeBERT as an adoption of computer vision techniques to produce efficient neural networks for downstream tasks. The authors concluded in their paper that the MobileBERT model achieved higher accuracy than other efficient networks such as DistilBERT [33] and BERT-PKD [52]. MobileBERT introduced two concepts into their NLP self-attention networks that are already in widespread use in CV neural networks:

1. Bottleneck layers: In RestNet [53], the 3x3 convolutions are computationally expensive, so a 1x1 "bottleneck" convolution is employed to reduce the number of channels input to each 3x3 convolution layer. Similarly, MobileBERT adopts bottleneck layers that reduce the number of channels before each self-attention layer, and this reduces the computational cost of the self-attention layers.

2. High-information flow residual connections. In BERT base architecture, the residual connections serve as links between the low-channel-count (768 channels) layers. The high-channel-count (3072 channels) layers in BERT base architecture do not have residual connections. However, the ResNet and Residual-SqueezeNet [54] CV networks connect the high-channel count layers with residuals, which enables higher information flow through the network. Similar to these CV networks, MobileBERT adds residual connections between the high-channel-count layers.

This proposed approach first trained a specially designed teacher model which is an inverted bottleneck incorporated with the BERT_Large model. Afterwards, a knowledge transfer technique was applied from this teacher to MobileBERT, empirical studies show that MobileBERT is 4.3x smaller and 5.5x faster than BERT base while achieving competitive results on GLUE metric. MobileBERT is designed to be as deep as BERT large while each layer is made much narrower via adopting bottleneck structure and balancing between self-attention and feed-forward networks.

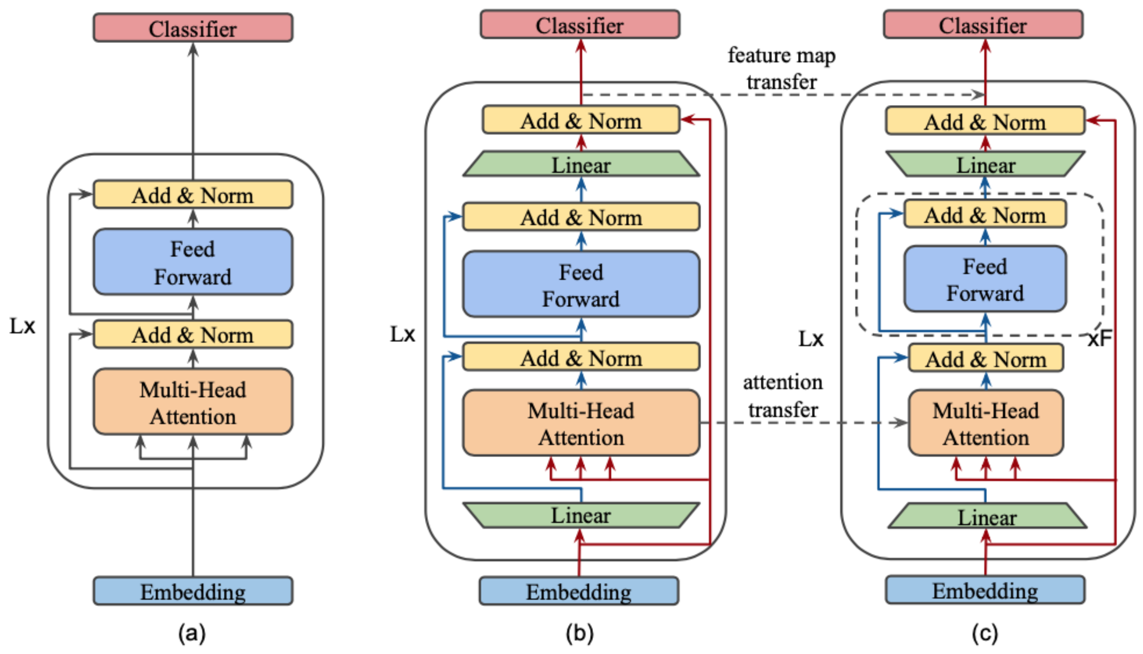


Figure 18: (a) BERT Architecture; (b) Inverted-Bottleneck (IB-BERT) Architecture; (c) MobileBERT Architecture

MobileBERT is trained by layer-to-layer imitating IB-BERT. We conduct knowledge transfer from IB-BERT to MobileBERT. Authors of MobileBERT have stated that the difference between their model and other “compressed” models is the stage in which they apply knowledge transfer.

(Jiao et al., 2019) proposed TinyBERT, a model that uses layer-wise distillation strategy for BERT in both pre-training and fine-tuning stages. **(Sanh et al., 2019)** proposed DistilBERT, which successfully halves the depth of BERT model by knowledge distillation in the pre-training stage and an optional fine-tuning stage. The key difference is that MobileBERT authors have used knowledge transfer only in the pre-training stage and do not require a fine-tuned teacher or data augmentation to train their model.
Previous work attempts to compress BERT by reducing its depth, while we focus on compressing BERT by reducing its width.

MobileBERT architecture is as deep as BERT_Large but each building block is made much smaller, the hidden dimension of each block is only 128 and introduces two linear transformations for each block to adjust its input and output dimensions to 512. We refer to such architecture as a bottleneck.
MobileBERT was trained by first constructing a teacher network and training it until convergence, afterwards, a knowledge transfer technique is applied from the teacher network to the student network which is MobileBERT in that case without the need to train it from scratch.

Stacked feed-forward networks were used in MobileBERT architecture since the bottleneck structure caused the imbalance problem. The broken balance is between the Multi-Head attention (MHA) module and the feed-forward network (FFN) module. MHA and FFN play different roles in the Transformer architecture: The former allows the model to jointly attend to information from different subspaces, while the latter increases the non-linearity of the model. In Original BERT, the ratio of the parameter numbers in MHA and FFN is always 1:2. But in the bottleneck structure, the inputs to the MHA are from wider feature maps (of inter-block size), while the inputs to the FFN are from narrower bottlenes (of intra-block size). This results in that the MHA modules in MobileBERT relatively contain more parameters. As a fix for this issue, the authors rebalance the relative size between MHA and FFN via stack feed-forward networks. In the figure

above, each MobileBERT layer contains one MHA but several stack FFN. In MobileBERT, they used 4 stacked FFN after each MHA.
When applying different training strategy for a new model, there exists three types of knowledge transfer:

1. Auxiliary Knowledge Transfer
2. Joint Knowledge Transfer
3. Progressive Knowledge Transfer

The conclusion of this proposed model is that bottleneck/inverted-bottleneck structures enable effective layer-wise knowledge transfer and that it was crucial to keep MobileBERT "deep" and "thin".

3.3 Available Datasets for Text Summarization

Currently there exists multiple datasets for abstractive and extractive text summarization which were collected from news outlets, scientific articles, customer reviews, and other forums. A model that generates a summary is called a generated summary or predicted summary. A human-made summary is referred to as a golden summary in this paper to distinguish the difference between the two summaries. Following section introduces a compiled list of datasets for extractive summarization.

1. CNN/Daily Mail Dataset

One of the most common datasets used for text summarization which was originally collected for Question Answering Systems [6]. It is a two-join dataset. First part is a dataset collected from a famous news channel called Cable News Network (CNN) that contains a total of 90,000 articles. Second part is a dataset collected from the Daily-Mail newspaper that contains a total of 197,000 articles. Each article in this large dataset has several bullet points that are marked as a golden-standard summary for this article. Such a format qualified it for extractive summarization and evaluating the metric based on the bullet points.

2. Document Understanding Conference (DUC 2002) Dataset

One of the earliest datasets collected for the text summarization. DUC 2002 was created by NIST (National Institute of Standards and Technology) for evaluation in the field of text summarization. DUC 2002 has 59 document sets with a total of 567 news articles, providing roughly 10 documents per set. Each document has two different gold summaries with a length of 100 words which gives more freedom to evaluate the article twice.

3. New York Times Dataset

A dataset collected by the famous New York times newspaper where the corpus consists of roughly 1.8 million articles [7]. 650,000 articles have been manually summarized which qualified it for extractive text summarization.

4. Opinosis Dataset

A dataset based on customer reviews on 51 various topics. Each topic in this dataset roughly has 100 documents each. It is originally collected from many websites that have review sections such as TripAdvisor for hotels or Amazon.com for various electronics [8].

5. GigaWord Dataset

The GigaWord dataset is a collection of four major sources of English newspapers [9].

- Associated Press World stream English Service
- Agence France Presse English Service
- The Xinhua News Agency English Service
- The New York Times Newswire Service

6. Blog Summarization Dataset

This dataset was created by collecting data from two blogs, Internet Explorer blog and Cosmic Variance [10]. The dataset is a random selection of 100 documents, 50 from each blog, with four golden summaries generated and ranked by human summarizers.

Chapter 4: Experiment Results

This chapter presents the results and the discussion of each experiment and concludes its output relevant to answering the research questions mentioned above in Part 1.4.

4.1 Preliminary Experiment - Studying the impact of different neural models on CNN/DM dataset

The objective of this experiment is to answer the first research question mentioned in Part 1.4, which is, how can a deep learning model improve summarization performance. In this preliminary experiment, three bert-base models were trained on CNN/DM dataset for evaluation, the first model was fine-tuned with a simple linear classifier, second model was fine-tuned with a recurrent neural network, and third model was fine-tuned with a small transformer network with 3 layers. The three models mentioned have been trained with 60,000 epochs where each epoch represents a complete pass on the entire training dataset, evaluated on ROUGE scores, ROUGE-1, ROUGE-2, and ROUGE-L. The experiment included training these models and saving their weights at a checkpoint of 10,000 epoch to construct Table 2 of the models' performance during the training phase. This experiment was conducted using a Google Colab Notebook utilizing the available GPU and the model training phase took around 6 days to complete given the constraint GPU usage sessions with dataset breakdown to train data (92%), validation data (4.3%), and testing data (3.7%).
The source code of this experiment can be found with documentation in Appendix B.1
In the first experiment with BERT+Classifier, the model produced the highest ROUGE scores at checkpoint 10,000 with *ROUGE-1 0.4276*, *ROUGE-2 0.19602*, and *ROUGE-L 0.38927* while the checkpoints afterwards did not improve and started degrading slowly as shown in the table below.

The results in Table 2 record the results of the evaluation algorithm that was defined in Chapter 2.7 where the summary is evaluated by computing the value of precision, recall, and F-score. Average_R represents the average weight of recall, Average_P represents the average weight of precision, and Average_F represents the average weight of F-measure.

BERT+Classifier Model		ROUGE-1	ROUGE-2	ROUGE-L
Model at 10,000	Average_R	0.54035	0.24896	0.49355
	Average_P	0.37125	0.17125	0.33967
	Average_F	**0.42576**	**0.19602**	**0.38927**
Model at 20,000	Average_R	0.52150	0.23837	0.47651
	Average_P	0.37436	0.17163	0.34265
	Average_F	0.42115	0.19255	0.38519
Model at 30,000	Average_R	0.51876	0.23671	0.47418
	Average_P	0.37627	0.17212	0.34448
	Average_F	0.42125	0.19218	0.38539
Model at 40,000	Average_R	0.49643	0.21839	0.45247
	Average_P	0.36751	0.16249	0.33569
	Average_F	0.40760	0.17954	0.37196
Model at 50,000	Average_R	0.50487	0.22603	0.46120
	Average_P	0.37147	0.16676	0.34009
	Average_F	0.41326	0.18503	0.37799

Table 4: Experiment on BERT+Classifier on 5 checkpoints

In the second experiment with BERT+Transformer, the model has outperformed BERT+Classifier from the first checkpoint and has shown promising results with comparison to it, and it scored the highest results of the three model variants with *ROUGE-1 0.43117, ROUGE-2 0.20152,* and *ROUGE-L 0.39542.*

BERT+Transformer Model		ROUGE-1	ROUGE-2	ROUGE-L
Model at 10,000	Average_R	0.53883	0.24859	0.49235
	Average_P	0.37254	0.17184	0.34099
	Average_F	0.42606	0.19624	0.38968
Model at 20,000	Average_R	0.53031	0.24455	0.48494
	Average_P	0.37748	0.17418	0.34573
	Average_F	0.42643	0.19644	0.39030
Model at 30,000	Average_R	0.51549	0.23547	0.47168
	Average_P	0.37774	0.17301	0.34639
	Average_F	0.42115	0.19238	0.38584
Model at 40,000	Average_R	0.52084	0.23732	0.47627
	Average_P	0.37388	0.17107	0.34262
	Average_F	0.42043	0.19168	0.38491

Model at 50,000	Average_R Average_P **Average_F**	0.53155 0.38415 **0.43117**	0.24820 0.18013 **0.20152**	0.48692 0.35261 **0.39542**

Table 5: Experiment on BERT+Transformer on 5 checkpoints

In the third experiment with BERT+RNN, the model has outperformed BERT+Classifier but did not outperform the BERT+Transformer model. However, the model showed high results during the first training checkpoints and produced the highest results at checkpoint 30,000 with scores *ROUGE-1 0.42846, ROUGE-2 0.19853,* and *ROUGE-L 0.39250*. The model did not improve in the checkpoints afterwards. Given the recorded values of the three models trained on the CNN/DM dataset, although the recurrent neural network architecture has less complexity than the transformer architecture, we conclude that the model BERT+Transformer was the best of the three variants at summary generation.

BERT+RNN Model		ROUGE-1	ROUGE-2	ROUGE-L
Model at 10,000	Average_R Average_P Average_F	0.53794 0.37356 0.42658	0.24791 0.17234 0.19645	0.49166 0.34194 0.39022
Model at 20,000	Average_R Average_P Average_F	0.53986 0.37361 0.42715	0.24986 0.17313 0.19752	0.49333 0.34193 0.39068
Model at 30,000	Average_R Average_P **Average_F**	0.52852 0.38177 **0.42846**	0.24485 0.17743 **0.19853**	0.48371 0.34999 **0.39250**
Model at 40,000	Average_R Average_P Average_F	0.51082 0.37088 0.41496	0.22932 0.16731 0.18657	0.46630 0.33929 0.37927
Model at 50,000	Average_R Average_P Average_F	0.52156 0.38065 0.42517	0.24001 0.17603 0.19587	0.47724 0.34899 0.38947

Table 6: Experiment on BERT+RNN on 5 checkpoints

4.2 Experiment 2 - Training DistillBERT Summarizer on CNN/DM dataset

This experiment shows an effort to explore the DistillBert model for extractive summarization tasks. I will fine-tune DistilBERT, a lite version of BERT model [33] and report the findings below.
DistilBERT was proposed in 2020 by ***Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf*** as a smaller, faster, cheaper, and lighter alternative to the bert base model when performing language understanding capabilities. The model is based on a technique called "Knowledge distillation" [37] which is a compression technique in which a compact model - the student - is trained to reproduce the behaviour of a larger model - the teacher - or an ensemble of models. DistilBERT has almost the same architecture as the BERT model except the token-type embeddings and the pooler are removed while the number of layers is reduced by a factor of 2. The authors have performed experiments with GLUE benchmark and concluded that DistilBERT retains 97% of BERT performance on downstream tasks. The motivation is to utilize this efficient compressed model for extractive summarization tasks and investigate whether DistilBERT can be a productionzed alternative with respect to the large BERT base model.

To run this experiment, I have used the publicly available single GPU provided by Google Colab to train this model which has taken around ~ 72 hours to complete. The model was trained on a total of 30,000 epochs and I used the ROUGE evaluation metrics to measure the performance and compare it with the benchmark bert baseline summarizer.
Libraries needed to install before running the experiment are:
torch==1.1.0 pytorch_pretrained_bert tensorboardX multiprocess transformers

DistillBERT Checkpoints		ROUGE-1	ROUGE-2	ROUGE-L
Model at 10,000	**Average_R** **Average_P** **Average_F**	**0.54046** **0.37102** **0.42535**	**0.24811** **0.17079** **0.19527**	**0.49321** **0.33920** **0.38856**
Model at 20,000	Average_R Average_P Average_F	0.52756 0.37355	0.24105 0.17152	0.48198 0.34193

		0.42238	0.19324	0.38631
Model at 30,000	Average_R Average_P Average_F	0.50242 0.36837 0.41015	0.22208 0.16356 0.18153	0.45819 0.33656 0.37443

Table 7: Experiment Results on Training DistilBERT on 3 checkpoints

During the training phase, the model generated a relatively higher score at checkpoint 10,000 when compared to the same model of different checkpoints. By computing the average of ROUGE-1, ROUGE-2, and ROUGE-L , distilbert summarizer generated competitive results that are slightly less when compared with the bert-base model in Table 6. Distilbert has ROUGE-1 score lesser than bert-base by 1.6%, and ROUGE-2 score lesser than bert-base by 3.5%, and ROUGE-L score lesser than bert-base by 1.9%. However, the number of parameters with DistilBert model is less than Bert model by ~36% which is directly proportional to the training time needed for these models. The observation here is that DistilBERT was able to retain approximately 98% of the traditional BERT model in terms of summarization performance with a decrease of ~36% in the number of trainable parameters. The params column, which is the sum of all weights and biases in a neural network, is recorded in Table 6 and can also be defined as the size of the trained model. The model's size is one of the important factors that need to be observed to reduce the training time and the summary generation of a summarizer model.

BERT Models	ROUGE-1	ROUGE-2	ROUGE-L	Params
bert-base	**43.23**	**20.24**	**39.63**	120.5 M
distilbert	42.54	19.53	38.86	**77.4 M**

Table 8: DistilBERT Performance with BERT baseline

4.3 Experiment 3 - Training SqueezeBERT Summarizer on CNN/DM dataset

This experiment shows an effort to explore the SqueezeBERT model, a recently developed model based on grouped convolutions instead of attention networks as an inspiration from computer vision research. I will fine-tune SqueezeBERT and report the findings below.

The motivation behind this experiment is to use SqueezeBERT where the authors have leveraged two ideas that are popular in the computer vision literature to accelerate NLP which are

1. Convolutions: replacement of the position-wise fully connected layers (PFC) with convolutions where the authors have stated that it has shown to produce significant speedups in computer vision networks.
2. Grouped Convolutions: incorporating grouped convolutions into self-attention networks.

SqueezeBERT was proposed in 2020 by ***Iandola, Forrest N., et al.*** as an architecture that is similar to BERT-base, but with the position-wise fully connected layers implemented as convolutions, and grouped convolutions for many of the layers. This model has been trained on a combination of Wikipedia and BooksCorpus [51] and in the experiment I will include the published model in huggingface library [55] and leverage SqueezeBERT for extractive summarization.

To run this experiment, I have used the publicly available single GPU provided by Google Colab to train this model which has taken around ~ 72 hours to complete. The model was trained on a total of 30,000 epochs and I used the ROUGE evaluation metrics to measure the performance and compare it with the benchmark bert baseline below in Table 7.

<u>Libraries needed to install before running the experiment are:</u>

torch==1.1.0 pytorch_pretrained_bert tensorboardX multiprocess transformers==4

SqueezeBERT Checkpoints		ROUGE-1	ROUGE-2	ROUGE-L
Model at 10,000	**Average_R**	**0.52760**	**0.24278**	**0.48227**
	Average_P	**0.37796**	**0.17427**	**0.34612**
	Average_F	**0.42538**	**0.19563**	**0.38922**
Model at 20,000	Average_R	0.50638	0.22715	0.46294
	Average_P	0.37708	0.16975	0.34537
	Average_F	0.41699	0.18717	0.38162
Model at 30,000	Average_R	0.48312	0.20655	0.43988
	Average_P	0.36130	0.15517	0.32963
	Average_F	0.39832	0.17044	0.36306

Table 9: Experiment Results on Training SqueezeBERT on 3 checkpoints

During the training phase, the model generated a relatively higher score at checkpoint 10,000 when compared to the same model of different checkpoints. By computing the average of ROUGE-1, ROUGE-2, and ROUGE-L, squeezebert summarizer generated competitive results that are slightly less when compared with the bert-base model. By observing the SqueezeBERT experiment, the model has generated ROUGE scores that are nearly identical or slightly better than the previous experiment with the DistilBERT model. This will be further discussed in the overall discussion. Based on Table 8 below, SqueezeBERT has ROUGE-1 score lesser than bert-base by 1.6%, and ROUGE-2 score lesser than bert-base by 3.35%, and ROUGE-L score lesser than bert-base by 1.8%. However, the number of parameters with DistilBert model is less than Bert model by ~ 49% which is directly proportional to the training time needed for these models. The observation here is that SqueezeBERT was able to retain approximately 98% of the traditional BERT model in terms of summarization performance with a decrease of ~ 49% in size. This experiment has demonstrated impressive results of training a summarizer with an architectural size that is almost half the size of the traditional bert base model while retaining 98% of its performance.

BERT Models	ROUGE-1	ROUGE-2	ROUGE-L	Params
bert-base	**43.23**	**20.24**	**39.63**	120.5 M
squeezebert	42.54	19.56	38.92	**62.13 M**

Table 10: SqueezeBERT Performance with BERT baseline

4.4 Experiment 4 - Training MobileBERT Summarizer on CNN/DM dataset

This experiment shows a comprehensive effort to train the MobileBERT model, a compressed architecture mainly targeted for low-resourced machines deployment. I will fine-tune MobileBERT and report the findings below. The SqueezeBERT model was released after the MobileBERT model as an inspiration for using computer vision techniques to advance research in the NLP domain. However, conducting an extractive summarization experiment with MobileBERT will help us further quantify the improvements of SqueezeBERT over it.

MobileBERT was proposed in 2020 by ***Zhiqing Sun, et al.,*** as a task-agnostic thinned version of the popular BERT model which has shown good results for downstream tasks with the advantage of being deployed to resource-limited mobile devices. MobileBERT is initially proposed not to outperform the performance of the BERT model, but rather to provide a compressed and accelerated version which can be deployed to mobile devices such as Google Pixel 4 phone [34].

The attempt of this experiment is to further train a smaller network than the previous model SqueezeBERT and observe its performance when the number of parameters have been significantly reduced. After training the model for 30,000 epochs as the previous experiment, MobileBERT produced relatively good ROUGE scores to previous experiments but became evidently lower than the baseline with ROUGE-1, ROUGE-2, and ROUGE-L. Based on the MobileBERT summarizer recorded in Table 9, the model has generated ROUGE-1 score lesser than bert-base by 6%, and ROUGE-2 score lesser than bert-base by 11%, and ROUGE-L score lesser than bert-base by ~ 7%. However, the number of parameters with MobileBERT model is less than Bert model by ~ 75% which is directly proportional to the training time needed for these models. The observation here in Table 9 is that MobileBERT was able to retain approximately 90-93% of the traditional BERT model in terms of summarization performance with a large decrease of ~ 75% in size. Although the experiment has demonstrated success in training a model with ¼ parameters of the traditional bert model, the performance slightly degraded and might be favored less than SqueezeBERT model since the performance retention was close to 98%. However, for applications that do not mind this reduced performance, this experiment is a great example of training a summarizer that can be deployed to low-resourced machines such as mobile devices and will still produce good summarization results.

BERT Models	ROUGE-1	ROUGE-2	ROUGE-L	Params
bert-base	**43.23**	**20.24**	**39.63**	120.5 M
mobilebert	40.59	17.98	36.99	**30.8 M**

Table 11: MobileBERT Performance with BERT baseline

Chapter 5: Conclusion

This chapter covers the overall discussion of the experiments conducted in the Methodology chapter with the main observations as well as the contribution of this thesis and possible areas of improvement as future work.

5.1 Experiments Discussion

After studying the summarization performance of different BERT models, there are some new insights that can be added to the literature as well as productionize a specific BERT-based summarization model given its competitive performance results recorded in Table 10. We have trained the BERT baseline and used it as a benchmark for what are known as "compressed models" such as DistilBERT, SqueezeBERT, and MobileBERT. Post-training these summarization models, they all retain > 90% performance of the traditional BERT model which has around 120.5 million parameters to train. The first experiment was an attempt to introduce a Distillation-based BERT model that has a lesser number of parameters by ~ 35%, and when testing the model, it retained ~ 98% of the BERT model. The second experiment was to train a more efficient model called SqueezeBERT which replaced all the attention layers with grouped convolutional layers for efficient computation. This experiment yielded an interesting observation that SqueezeBERT maintains the same performance as DistilBERT with uni-grams and slightly better with bi-grams and longest common sequences. SqueezeBERT retains 98% performance of the traditional BERT with even fewer parameters by 49% consisting of 62.13 million parameters instead of the traditional BERT with 120.5 million parameters. SqueezeBERT is a good candidate for training a summarizer with nearly half the size of the original model with minimal downgrade in the summarization performance. It also gave a key observation that using efficient networks inspired by computer vision literature such as grouped convolutional layers can improve NLP downstream tasks, and for this thesis, it improved the summarization task. The MobileBERT experiment was an attempt to further reduce the size of the summarizer given the encouragement from the SqueezeBERT experiment. The observation from that experiment is that

MobileBERT lost some summarization performance, however, the number of its parameters has significantly reduced, making the architecture nearly ¼ the size of the traditional BERT model, which was inspired by MobileBERT authors to use this model for low-resourced devices. The conclusion from this experiment is that MobileBERT can be used to deploy a miniature summarizer to mobile devices and still perform good summarization results retaining ~ 90% of the traditional BERT model which can be a good trade-off.

BERT Models	ROUGE-1	ROUGE-2	ROUGE-L	Params
bert-base	43.23	20.24	39.63	120.5 M
distilbert	42.54	19.53	38.86	77.4 M
squeezebert	42.54	19.56	38.92	62.13 M
mobilebert	40.59	17.98	36.99	30.8 M

Table 12: Comprehensive Extractive Summarization Performance

Post-training these models with their validation scores, their final testing scores in the following format (ROUGE-1/ROUGE-2/ROUGE-L) are; DistilBERT (42.308/19.302/38.637), SqueezeBERT (42.325/19.341/38.710), and MobileBERT (40.288/17.618/36.412).

The table below shows the testing time of three document examples that were used as part of the experiments' observation. Refer to Appendix A for more details of these examples. Each pre-trained summarizer has generated an extractive summary from a URL to a news article from CNN website. The data was first preprocessed and cleaned before passing it to the summarizer. The first example produced a clean document of word count approximately 300 while the second example produced approximately 700 words and the third example had a smaller word count to 250. These examples were tested on a Google Colab Notebook where I first cloned my own forked repository that loads my pre-trained models and generates summaries. Using Python Profile, I was able to record the time taken to finish executing the summarize() method. However the testing time might not be the most accurate, but they give a rough estimate that bert-base takes longer to generate a summary while mobilebert takes the least amount of time to generate a summary. DistilBERT and SqueezeBERT both generate nearly similar summaries with SqueezeBERT taking relatively less time to complete the task.

BERT Models	Example A (seconds)	Example B (seconds)	Example C (seconds)
bert-base	2.85150	3.70764	2.96757
DistilBERT	2.68558	2.91017	2.55496
SqueezeBERT	2.58510	2.89347	2.42758
MobileBERT	**2.37579**	**2.54992**	**2.31980**

Table 13: Testing Time of different Models on three document examples.

Based on the experiments conducted above, one key observation here is that distillation approaches can be effective in improving text summarization performance. The term "distillation" is popularly used in efficient NLP networks and the process of distillation is having a "student" network trained to replicate a "teacher" network. In literature, some researchers distill only the final layer of a neural network while others distill the hidden layers as well [37]. Using SqueezeBERT model for text summarization has shown that the number of parameters in the architecture can be significantly reduced by replacing self-attention networks with convolutional layers while maintaining a competitive performance with other compressed models such as DistilBERT, MobileBERT, and also with the original model, BERT-base.

5.2 Contribution

After completing the experiments, there is a potential productionized version of SqueezeBERT extractive summarizer from the results recorded. SqueezeBERT has fewer parameters than DistilBERT by approximately 20% and yields the same ROUGE-1 score and slightly higher in ROUGE-2 and ROUGE-L score. Although SqueezeBERT and DistilBERT produce slightly lower scores in the BERT baseline model, SqueezeBERT has an advantage of lesser training time and fewer parameters than baseline model by ~ 48.44% which is close to half the BERT baseline model's trained parameters.

In this thesis, we were able to train different summarization models to answer the research question stated above of which BERT variant can perform better in summarization and produce a study in an effort to expand the literature review by comparing different models in terms of summary generation accuracy and the model's complexity. We also contribute with the

“SqueezeBERTSumm” model trained for text summarization with retained performance by 98% of the original BERT model and 49% less trainable parameters. The model is also provided with the source code for future research and development.

5.3 Future Work

These experiments were conducted on a single GPU resource from Google Colab and although it was sufficient for the experiments above, there is some room for additional work such as hyperparameter tuning to these pretrained models in an attempt to generate better summarization performance. Another possible future work is to train these models on domain-specific datasets and produce an extractive summarizer dedicated to the use case such as medical or academic extractive summarizer. Further possible future work is exploring the potential of fine tuning the SqueezeBERT model for abstractive summarization instead of extractive summarization and report the performance results if there are any significant findings.

To further reduce the size of the pretrained model, there is also room for adopting techniques such as quantization and pruning.

1. Quantization is a family of techniques which aims to reduce the number of bits required to store each parameter and/or activation in a neural network, while at the same time maintaining the accuracy of that network. This technique has been applied in different works of NLP [56][57].
2. Pruning aims to directly eliminate certain parameters from the network while also maintaining accuracy, thereby reducing the storage and potentially computational cost of that network; for an application of this NLP, this research demonstrates an application of pruning [58].

6. References

1. M. Gambhir and V . Gupta, “Recent automatic text summarization techniques: a survey,” Artificial Intelligence Review, vol. 47, no. 1, pp. 1–66, 2017
2. Luhn, H. P. (1958). The automatic creation of literature abstracts. IBM Journal of Research Development, 2(2), 159–165.
3. Nallapati R, Zhai F, Zhou B (2017) Summarunner: a recurrent neural network-based sequence model for extractive summarization of documents. In: Proc. Thirty-First AAAI Conference on Artificial Intelligence, pp 3075–3081.
4. Eduard Hovy and Chin Yew Lin, Automated text summarization in SUMMARIST, MIT Press, 1999, pages 81–94.
5. Babar, S. A., & Patil, P. D. (2015). Improving performance of text summarization.
6. Hermann, K.M.; Kocisky ́, T.; Grefen- stette, E.; Espeholt, L.; Kay, W.; Suleyman, M.; and Blun- som, P. 2015. Teaching machines to read and comprehend. CoRR abs/1506.03340.
7. Sandhaus, Evan. The New York Times Annotated Corpus LDC2008T19. DVD. Philadelphia: Linguistic Data Consortium, 2008. Available: https://catalog.ldc.upenn.edu/LDC2008T19
8. Ganesan, K., Zhai, C., & Han, J. (2010). Opinosis: A graph-based approach to abstractive summarization of highly redundant opinions. Proceedings of the 23rd international conference on computational linguistics (pp. 340–348).
9. Graff, David, and Christopher Cieri. English Gigaword LDC2003T05. Web Download. Philadelphia: Linguistic Data Consortium, 2003. Available: https://catalog.ldc.upenn.edu/LDC2003T05

10. Hu, M., Sun, A., & Lim, E.-P. (2007). Comments-oriented blog summarization by sentence extraction. In Proceedings of the 16th ACM conference on information and knowledge management (pp. 901–904).
11. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed represen- tations of words and phrases and their compositionality. In Advances in neural information processing systems, 3111– 3119.
12. Mihalcea, R., & Tarau, P. (2004). TextRank: Bringing order into texts. In Proceedings of empirical methods on natural language processing (pp. 404– 411).
13. Erkan, G., & Radev, D. (2004). Lexrank: Graph-based lexical centrality as salience in text summarization. Journal of Artificial Intelligence Research, 22(1), 457–479.
14. Steinberger, J., & Jezek, K. (2004). Using latent semantic analysis in text summarization and summary evaluation. In Proceedings of 7th International Conference on Information System Implementation and Modeling (pp. 93–100).
15. Nenkova, A., & Vanderwende, L. (2005). The impact of frequency on summarization. Technical report, Microsoft Research (MSR-TR-2005-101).
16. Chung, J.; Gu ̈lc ̧ehre, C ̧.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated re- current neural networks on sequence modeling. CoRR abs/1412.3555.
17. Cheng, J., and Lapata, M. 2016. Neural summarization by extracting sentences and words. 54th Annual Meeting of the Association for Computational Linguistics.
18. Joshi, Akanksha & Fidalgo, Eduardo & Alegre, Enrique & Fernández-Robles, Laura. (2019). SummCoder: An unsupervised framework for extractive text summarization based on deep auto-encoders. Expert Systems with Applications. 129. 200-215. 10.1016/j.eswa.2019.03.045.
19. Lin, C. Y. (2004). Rouge: A package for automatic evaluation of summaries. In Proceedings of the workshop on text summarization branches out (WAS 2004) (pp. 25–26).
20. S. P. Singh, A. Kumar, A. Mangal and S. Singhal, "Bilingual automatic text summarization using unsupervised deep learning," 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), Chennai, 2016, pp. 1195-1200.
21. S. Hochreiter, “Long short-term memory,” 1997. Neural computation.

22. K. C. et al., “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” 2014. arXiv:1406.1078v3.
23. Pennington, Jeffrey, et al. “GloVe: Global Vectors for Word Representation.” ACL Anthology, www.aclweb.org/anthology/D14-1162/.
24. Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018, March 22). Deep contextualized word representations. Retrieved January 03, 2021, from https://arxiv.org/abs/1802.05365v2
25. Devlin, Jacob, et al. “BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding.” ArXiv.org, 24 May 2019, arxiv.org/abs/1810.04805.
26. Liu, Yang and Mirella Lapata. “Text Summarization with Pretrained Encoders.” EMNLP/IJCNLP (2019).
27. Yue Dong, Yikang Shen, Eric Crawford, Herke van Hoof, and Jackie Chi Kit Cheung. 2018. Banditsum: Extractive summarization as a contextual bandit. InProceedings of the EMNLP Conference.
28. Qingyu Zhou, Nan Yang, Furu Wei, Shaohan Huang, Ming Zhou, and Tiejun Zhao. 2018. Neural docu- ment summarization by jointly learning to score and select sentences. In Proceedings of the ACL Confer- ence.
29. Xu, Jiacheng, et al. “Discourse-Aware Neural Extractive Text Summarization.” ArXiv.org, 25 Apr. 2020, arxiv.org/abs/1910.14142.
30. Zhong, Ming, et al. “Extractive Summarization as Text Matching.” ACL Anthology, www.aclweb.org/anthology/2020.acl-main.552/.
31. Liu, Yang. “Fine-Tune BERT for Extractive Summarization.” ArXiv.org, 5 Sept. 2019, arxiv.org/abs/1903.10318.
32. Iandola, Forrest N., et al. “SqueezeBERT: What Can Computer Vision Teach NLP about Efficient Neural Networks?” ArXiv.org, 19 June 2020, arxiv.org/abs/2006.11316.
33. Victor Sanh, Lysandre Debut, Julien Chaumond, Thomas Wolf. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. 1 Mar 2020, https://arxiv.org/pdf/1910.01108v4.pdf
34. Zhiqing Sun, et al. “MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices”. 6 April 2020, https://arxiv.org/pdf/2004.02984.pdf

35. Lucy Lu Wang, Kyle Lo, Yoganand Chandrasekhar, Russell Reas, Jiangjiang Yang, Darrin Eide, Kathryn Funk, Rodney Kinney, Ziyang Liu, William Merrill, Paul Mooney, Dewey Murdick, Devvret Rishi, Jerry Sheehan, Zhihong Shen, Brandon Stilson, Alex D. Wade, Kuansan Wang, Chris Wilhelm, Boya Xie, Douglas Raymond, Daniel S. Weld, Oren Etzioni, and Sebastian Kohlmeier. Cord-19: The covid-19 open research dataset, 2020.
36. Cristian Bucila, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In KDD, 2006.
37. Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. ArXiv, abs/1503.02531, 2015.
38. F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size," arXiv:1602.07360, 2016.
39. K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," Biological Cybernetics, 1980.
40. Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," Neural Computation, 1989.
41. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolu-
tional Neural Networks," in NeurIPS, 2012.
42. A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applica- tions," arXiv:1704.04861, 2017.
43. X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in CVPR, 2018.
44. M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in International Conference on Machine Learning (ICML), 2019. [Online]. Available: http://proceedings.mlr.press/v97/tan19a.html

45. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in Conference of the North American Chapter of the Association for Computational Linguistics (NAACL), 2019.
46. A.Vaswani,N.Shazeer,N.Parmar,J.Uszkoreit,L.Jones,A.N.Gomez,L.Kaiser,andI.Polo-sukhin, “Attention is all you need,” in Conference on Neural Information Processing Systems (NeurIPS), 2017.
47. A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, “Improving language understanding by generative pre-training,”https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_pap 2018.
48. K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, “ELECTRA: Pre-training text encoders as discriminators rather than generators,” in International Conference on Learning Represen- tations (ICLR), 2020.
49. Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, “RoBERTa: A robustly optimized bert pretraining approach,” arXiv:1907.11692, 2019.
50. Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, “Google’s neural machine trans- lation system: Bridging the gap between human and machine translation,” arXiv:1609.08144, 2016.
51. Y.Zhu,R.Kiros,R.Zemel,R.Salakhutdinov,R.Urtasun,A.Torralba,andS.Fidler,“Aligning books and movies: Towards story-like visual explanations by watching movies and reading books,” in IEEE International Conference on Computer Vision (ICCV), 2015.
52. Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for bert model com- pression. arXiv preprint arXiv:1908.09355.
53. K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

54. F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size," arXiv:1602.07360, 2016.
55. https://huggingface.co
56. S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Q- BERT: Hessian based ultra low precision quantization of bert," in AAAI, 2020.
57. O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat, "Q8BERT: Quantized 8bit bert," arXiv:1910.06188, 2019.
58. V. Sanh, T. Wolf, and A. M. Rush, "Movement pruning: Adaptive sparsity by fine-tuning," arXiv:2005.07683, 2020.
59. Kieuvongngam, V., Tan, B., & Niu, Y. (2020). Automatic Text Summarization of COVID-19 Medical Research Articles using BERT and GPT-2. ArXiv, abs/2006.01997.
60. Tretyak, V., & Stepanov, D. (2020). Combination of abstractive and extractive approaches for summarization of long scientific texts. ArXiv, abs/2006.05354.
61. https://github.com/nlpyang/PreSumm
62. Ya-Han Hu, Yen-Liang Chen, Hui-Ling Chou, Opinion mining from online hotel reviews – A text summarization approach, Information Processing & Management, Volume 53, Issue 2, 2017, Pages 436-449, ISSN 0306-4573, https://doi.org/10.1016/j.ipm.2016.12.002.
63. Shixia Liu, Michelle X. Zhou, Shimei Pan, Yangqiu Song, Weihong Qian, Weijia Cai, and Xiaoxiao Lian. 2012. TIARA: Interactive, Topic-Based Visual Text Summarization and Analysis. <i>ACM Trans. Intell. Syst. Technol.</i> 3, 2, Article 25 (February 2012), 28 pages. DOI:https://doi.org/10.1145/2089094.2089101
64. Yogesh Sankarasubramaniam, Krishnan Ramanathan, Subhankar Ghosh, Text summarization using Wikipedia, Information Processing & Management, Volume 50, Issue 3, 2014, Pages 443-461, ISSN 0306-4573, https://doi.org/10.1016/j.ipm.2014.02.001.
65. Changjian Fang, Dejun Mu, Zhenghong Deng, Zhiang Wu, Word-sentence co-ranking for automatic extractive text summarization, Expert Systems with Applications, Volume 72, 2017, Pages 189-195, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2016.12.021.

66. Al-Saleh, A.B., Menai, M.E.B. Automatic Arabic text summarization: a survey. Artif Intell Rev 45, 203–234 (2016). https://doi-org.libproxy.aucegypt.edu/10.1007/s10462-015-9442-x
67. Wafaa S. El-Kassas, Cherif R. Salama, Ahmed A. Rafea, Hoda K. Mohamed, EdgeSumm: Graph-based framework for automatic text summarization, Information Processing & Management, Volume 57, Issue 6, 2020, 102264, ISSN 0306-4573, https://doi.org/10.1016/j.ipm.2020.102264.
68. Cilibrasi, R. L., & Vitanyi, P. M. (2007). The google similarity distance. IEEE Transactions on Knowledge and Data Engineering, 19(3), 370–383.
69. J. N. Madhuri and R. Ganesh Kumar, "Extractive Text Summarization Using Sentence Ranking," 2019 International Conference on Data Science and Communication (IconDSC), 2019, pp. 1-3, doi: 10.1109/IconDSC.2019.8817040.
70. Belwal, R.C., Rai, S. & Gupta, A. A new graph-based extractive text summarization using keywords or topic modeling. J Ambient Intell Human Comput 12, 8975–8990 (2021). https://doi.org/10.1007/s12652-020-02591-x

7. Appendix

Appendix A: Examples of Extractive Summaries

A.1 Example 1

News Article URL:
https://edition.cnn.com/world/live-news/coronavirus-pandemic-vaccine-updates-05-28-21/index.html

[Document]:
The Equal Employment Opportunity Commission said Friday that employers are not legally prohibited from offering incentives to employees to get vaccinated for Covid-19, and that companies administering vaccines themselves can also do so "as long as the incentives are not coercive." The EEOC already said in December that companies can legally mandate all employees re-entering the workplace and new hires be vaccinated for Covid-19. But there are two exemptions companies must allow for, according to the EEOC: a disability or religious reasons. In its updated guidance released Friday, the EEOC now says employers are permitted to offer incentives to employees who voluntarily provide information they've been vaccinated by a third party — and that there is no limit to the size of those incentives. For example, workers at both McDonalds corporate headquarters and restaurants will get up to four hours of paid time off to get vaccinated.

Employees at Bolthouse Farms will get a $500 bonus if vaccinated. And at Kroger, employees will get a one-time payment of $100 if they show proof of vaccination. The agency adds that if employers obtain employee vaccination information, however, they “must keep vaccination information confidential.” Also, if the employer plans to administer the vaccine itself, incentives must not be large enough to be considered "coercive." "Because vaccinations require employees to answer pre-vaccination disability-related screening questions," the EEOC states, "a very large incentive could make employees feel pressured to disclose protected medical information." Employers are also legally permitted to provide employees and their family members with educational information about Covid-19 vaccines, raise awareness about the benefits, and address common questions and concerns.

[Bert Base - Summary]:
The Equal Employment Opportunity Commission said Friday that employers are not legally prohibited from offering incentives to employees to get vaccinated for Covid-19, and that companies administering vaccines themselves can also do so "as long as the incentives are not coercive.” The EEOC already said in December that companies can legally mandate all employees re-entering the workplace and new hires be vaccinated for Covid-19. But there are two exemptions companies must allow for, according to the EEOC: a disability or religious reasons. For example, workers at both McDonalds corporate headquarters and restaurants will get up to four hours of paid time off to get vaccinated.

[DistilBERT - Summary]:
The Equal Employment Opportunity Commission said Friday that employers are not legally prohibited from offering incentives to employees to get vaccinated for Covid-19, and that companies administering vaccines themselves can also do so "as long as the incentives are not coercive.” The EEOC already said in December that companies can legally mandate all employees re-entering the workplace and new hires be vaccinated for Covid-19. But there are two exemptions companies must allow for, according to the EEOC: a disability or religious reasons. For example, workers at both McDonalds corporate headquarters and restaurants will get up to four hours of paid time off to get vaccinated.

[SqueezeBERT - Summary]:
The Equal Employment Opportunity Commission said Friday that employers are not legally prohibited from offering incentives to employees to get vaccinated for Covid-19, and that companies administering vaccines themselves can also do so "as long as the incentives are not coercive.” The EEOC already said in December that companies can legally mandate all employees re-entering the workplace and new hires be vaccinated for Covid-19. But there are two exemptions companies must allow for, according to the EEOC: a disability or religious reasons. "Because vaccinations require employees to answer pre-vaccination disability-related screening questions," the EEOC states, "a very large incentive could make employees feel pressured to disclose protected medical information."

[MobileBERT - Summary]:
But there are two exemptions companies must allow for, according to the EEOC: a disability or religious reasons. The Equal Employment Opportunity Commission

said Friday that employers are not legally prohibited from offering incentives to employees to get vaccinated for Covid-19, and that companies administering vaccines themselves can also do so "as long as the incentives are not coercive." The EEOC already said in December that companies can legally mandate all employees re-entering the workplace and new hires be vaccinated for Covid-19. For example, workers at both McDonalds corporate headquarters and restaurants will get up to four hours of paid time off to get vaccinated.

A.2 Example 2

News Article URL: https://edition.cnn.com/travel/article/australia-international-travel-borders-intl-hnk/index.html

[Document]:
(CNN) — Australian Prime Minister Scott Morrison on Friday, October 1 announced plans to reopen the country's borders to fully vaccinated citizens and permanent residents, moving from a strict zero-Covid strategy to a model of living with the virus. The move -- expected to take effect in November -- comes more than 18 months after Australia closed its borders to the world in response to the pandemic , imposing quotas on arrivals that left thousands of citizens stranded overseas and mandatory 14-day hotel quarantine for those able to make the journey. Morrison also said that with 55% of the country now fully vaccinated and first dose rates approaching 80%, the government "has been finalizing plans so Australian families can be reunited, Australian workers can travel in and out of our country, and we can work towards welcoming tourists back to our shores." "Many countries around the world have now safely reopened to international travel and it will shortly be time for Australia to take the next step," he added. Related content Travel to Australia during Covid-19: What you need to know before you go Australia's tough restrictions, combined with localized measures including lockdowns , have helped the country mostly control Covid-19 inside its borders. To date, Australia has reported more than 107,000 confirmed cases of the virus and 1,311 deaths, according to data from Johns Hopkins University. But as states and territories have struggled to contain outbreaks of the highly-contagious Delta variant in recent months, the Australian government said it had been forced to reexamine its zero-Covid approach. Under the plan outlined by Morrison Friday, Australians and permanent residents would be allowed to quarantine at home -- which would essentially remove the caps

limiting the number of people allowed into the country. Those travelers would also only face seven days of quarantine. The move was "anticipated" to follow the completion of home quarantine trials in two states, Morrison said. "To reopen safely and to stay safely open, firstly we need home quarantine pilots in New South Wales and South Australia to conclude and be successful so they can role out at scale." For those not vaccinated with an approved shot, 14 days' quarantine in a government managed facility will still be in place. Related content Australian ad showing Covid patient gasping for air sparks backlash as country battles Delta variant Australia's Therapeutic Goods Administration had previously approved four vaccines -- made by Pfizer, AstraZeneca, Moderna and Johnson and Johnson. Morrison said Friday that two more had been added to the list as of Friday -- China's Sinovac and India's Covishield. "Australian citizens and permanent residents who cannot be vaccinated -- for example if they are under 12 or have a medical condition -- will be treated as vaccinated for the purposes of their travel," a statement from the government added. Without naming an exact date, Morrison estimated that an 80% fully vaccinated target set by the government as a condition for the border changes to come into effect was likely to be reached sometime in November. Thousands of Australians have been stranded overseas for months due to the country's tough border conditions, which has led to huge frustration among citizens who felt their country had abandoned them. Last Christmas, an estimated 39,000 Australian residents were not able to make it back into the country to celebrate the holiday. While the new border opening will allow them to return, Morrison was less clear on when people who are not citizens or permanent residents will be able to visit the country. "We will also work towards completely quarantine-free travel for certain countries, such as New Zealand , when it is safe to do so," Morrison said Friday. But Jennifer Evans, chief executive officer of the National Association of Testing Authorities, said she was waiting to hear how the government planned to handle Covid-19 testing for incoming travelers, given even vaccinated people can carry the disease. "Right now there's nothing mandated by the federal government. I'm asking that we follow the UK government's example and require accreditation for all labs and facilities doing pre-travel testing," she said in a statement.

[Bert Base - Summary]:

(CNN) — Australian Prime Minister Scott Morrison on Friday, October 1 announced plans to reopen the country's borders to fully vaccinated citizens and permanent residents, moving from a strict zero-Covid strategy to a model of living with the virus. The move -- expected to take effect in November -- comes more than 18 months after Australia closed its borders to the world in response to the pandemic , imposing quotas on arrivals that left thousands of citizens stranded overseas and mandatory 14-day hotel quarantine for those able to make the journey. But as states and territories have struggled to contain outbreaks of the highly-contagious Delta variant in recent months, the Australian government said it had been forced to reexamine its zero-Covid approach.

[DistilBERT - Summary]:

(CNN) — Australian Prime Minister Scott Morrison on Friday, October 1 announced plans to reopen the country's borders to fully vaccinated citizens and permanent residents, moving from a strict zero-Covid strategy to a model of living with the virus. The move -- expected to take effect in November -- comes more than 18 months after Australia closed its borders to the world in response to the pandemic , imposing quotas on arrivals that left thousands of citizens stranded overseas and mandatory 14-day hotel quarantine for those able to make the journey. To date, Australia has reported more than 107,000 confirmed cases of the virus and 1,311 deaths, according to data from Johns Hopkins University.

[SqueezeBERT - Summary]:
(CNN) — Australian Prime Minister Scott Morrison on Friday, October 1 announced plans to reopen the country's borders to fully vaccinated citizens and permanent residents, moving from a strict zero-Covid strategy to a model of living with the virus. The move -- expected to take effect in November -- comes more than 18 months after Australia closed its borders to the world in response to the pandemic , imposing quotas on arrivals that left thousands of citizens stranded overseas and mandatory 14-day hotel quarantine for those able to make the journey. To date, Australia has reported more than 107,000 confirmed cases of the virus and 1,311 deaths, according to data from Johns Hopkins University.

[MobileBERT - Summary]:
(CNN) — Australian Prime Minister Scott Morrison on Friday, October 1 announced plans to reopen the country's borders to fully vaccinated citizens and permanent residents, moving from a strict zero-Covid strategy to a model of living with the virus. The move -- expected to take effect in November -- comes more than 18 months after Australia closed its borders to the world in response to the pandemic , imposing quotas on arrivals that left thousands of citizens stranded overseas and mandatory 14-day hotel quarantine for those able to make the journey. Morrison also said that with 55% of the country now fully vaccinated and first dose rates approaching 80%, the government "has been finalizing plans so Australian families can be reunited, Australian workers can travel in and out of our country, and we can work towards welcoming tourists back to our shores."

A.3 Example 3

News Article URL: https://www.cnn.com/2020/05/29/tech/facebook-violence-trump/index.html

[Document]:
(CNN) Over and over again in 2018, during an apology tour that took him from the halls of the US Congress to an appearance before the European Parliament, Mark Zuckerberg said Facebook had failed to "take a broad enough view of our responsibilities." But two years later, Zuckerberg and Facebook are still struggling with their responsibilities and how to handle one of their most famous users: President Donald Trump. Despite Zuckerberg having previously indicated any post that "incites violence" would be a line in the sand — even if it came from a politician — Facebook remained silent for hours Friday after Trump was accused of glorifying violence in posts that appeared on its platforms. At 12:53am ET on Friday morning, as cable news networks carried images of fires and destructive protests in Minneapolis, the President tweeted : "These THUGS are dishonoring the memory of George Floyd, and I won't let that happen. Just spoke to Governor Tim Walz and told him that the Military is with him all the way. Any difficulty and we will assume control but, when the looting starts, the shooting starts. Thank you!" His phrase "when the looting starts,

the shooting starts," mirrors language used by a Miami police chief in the late 1960s in the wake of riots. Its use was immediately condemned by a wide array of individuals, from historians to members of rival political campaigns. Former Vice President and presumptive Democratic nominee Joe Biden said Trump was "calling for violence against American citizens during a moment of pain for so many."

[Bert Base - Summary]:
Despite Zuckerberg having previously indicated any post that "incites violence" would be a line in the sand — even if it came from a politician — Facebook remained silent for hours Friday after Trump was accused of glorifying violence in posts that appeared on its platforms. But two years later, Zuckerberg and Facebook are still struggling with their responsibilities and how to handle one of their most famous users: President Donald Trump. (CNN) Over and over again in 2018, during an apology tour that took him from the halls of the US Congress to an appearance before the European Parliament, Mark Zuckerberg said Facebook had failed to "take a broad enough view of our responsibilities."

[DistilBERT - Summary]:
But two years later, Zuckerberg and Facebook are still struggling with their responsibilities and how to handle one of their most famous users: President Donald Trump. (CNN) Over and over again in 2018, during an apology tour that took him from the halls of the US Congress to an appearance before the European Parliament, Mark Zuckerberg said Facebook had failed to "take a broad enough view of our responsibilities." Despite Zuckerberg having previously indicated any post that "incites violence" would be a line in the sand — even if it came from a politician — Facebook remained silent for hours Friday after Trump was accused of glorifying violence in posts that appeared on its platforms.

[SqueezeBERT - Summary]:
But two years later, Zuckerberg and Facebook are still struggling with their responsibilities and how to handle one of their most famous users: President Donald Trump. (CNN) Over and over again in 2018, during an apology tour that took him from the halls of the US Congress to an appearance before the European Parliament, Mark Zuckerberg said Facebook had failed to "take a broad enough view of our responsibilities." Despite Zuckerberg having previously indicated any post that "incites violence" would be a line in the sand — even if it came from a politician — Facebook remained silent for hours Friday after Trump was accused of glorifying violence in posts that appeared on its platforms.

[MobileBERT - Summary]:
(CNN) Over and over again in 2018, during an apology tour that took him from the halls of the US Congress to an appearance before the European Parliament, Mark Zuckerberg said Facebook had failed to "take a broad enough view of our responsibilities." But two years later, Zuckerberg and Facebook are still struggling with their responsibilities and how to handle one of their most famous users: President Donald Trump. Despite Zuckerberg having previously indicated any post that "incites violence" would be a line in the sand — even if it came from a politician — Facebook remained silent for hours Friday after Trump was accused of glorifying violence in posts that appeared on its

platforms.

Appendix B: Source Code

B.1 Modified Forked Repository from PreSumm paper

https://github.com/ShehabMMohamed/PreSumm

B.2 Demo Source Code

https://colab.research.google.com/drive/1M42XEWU09lGRmBtmdPQ8JG-hfrl14a2J?usp=sharing

B.3 Pretrained BERT Models

https://drive.google.com/drive/folders/1mtMCGyM-NLKowjtM5U3EQ959JbgYLmTV?usp=sharing

Appendix C: SqueezeBERT Model Rouge Scores

```
---------------------------------------------
Rouges at step 10000
ROUGE-F(1/2/3/l): 42.54/19.56/38.92
ROUGE-R(1/2/3/l): 52.76/24.28/48.23
---------------------------------------------
Rouges at step 20000
ROUGE-F(1/2/3/l): 41.70/18.72/38.16
ROUGE-R(1/2/3/l): 50.64/22.71/46.29
---------------------------------------------
Rouges at step 30000
ROUGE-F(1/2/3/l): 39.83/17.04/36.31
ROUGE-R(1/2/3/l): 48.31/20.66/43.99
---------------------------------------------
```

Appendix D: SqueezeBERT Extractive Summarizer Architecture

```
ExtSummarizer(
  (bert): Bert(
    (model): SqueezeBertModel(
      (embeddings): SqueezeBertEmbeddings(
        (word_embeddings): Embedding(30528, 768, padding_idx=0)
        (position_embeddings): Embedding(512, 768)
        (token_type_embeddings): Embedding(2, 768)
        (LayerNorm): LayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1)
      )
      (encoder): SqueezeBertEncoder(
        (layers): ModuleList(
(0): SqueezeBertModule(
  (attention): SqueezeBertSelfAttention(
    (query): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (key): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (value): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (dropout): Dropout(p=0.1)
    (softmax): Softmax()
    (matmul_qk): MatMulWrapper()
    (matmul_qkv): MatMulWrapper()
  )
  (post_attention): ConvDropoutLayerNorm(
    (conv1d): Conv1d(768, 768, kernel_size=(1,), stride=(1,))
    (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1)
  )
  (intermediate): ConvActivation(
    (conv1d): Conv1d(768, 3072, kernel_size=(1,), stride=(1,), groups=4)
  )
  (output): ConvDropoutLayerNorm(
    (conv1d): Conv1d(3072, 768, kernel_size=(1,), stride=(1,), groups=4)
    (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1)
  )
)
(1): SqueezeBertModule(
  (attention): SqueezeBertSelfAttention(
    (query): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (key): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (value): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (dropout): Dropout(p=0.1)
    (softmax): Softmax()
    (matmul_qk): MatMulWrapper()
    (matmul_qkv): MatMulWrapper()
  )
  (post_attention): ConvDropoutLayerNorm(
    (conv1d): Conv1d(768, 768, kernel_size=(1,), stride=(1,))
    (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1)
  )
  (intermediate): ConvActivation(
    (conv1d): Conv1d(768, 3072, kernel_size=(1,), stride=(1,), groups=4)
  )
  (output): ConvDropoutLayerNorm(
    (conv1d): Conv1d(3072, 768, kernel_size=(1,), stride=(1,), groups=4)
    (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1)
  )
)
(2): SqueezeBertModule(
  (attention): SqueezeBertSelfAttention(
    (query): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (key): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (value): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (dropout): Dropout(p=0.1)
    (softmax): Softmax()
    (matmul_qk): MatMulWrapper()
    (matmul_qkv): MatMulWrapper()
  )
  (post_attention): ConvDropoutLayerNorm(
    (conv1d): Conv1d(768, 768, kernel_size=(1,), stride=(1,))
    (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1)
  )
  (intermediate): ConvActivation(
    (conv1d): Conv1d(768, 3072, kernel_size=(1,), stride=(1,), groups=4)
  )
  (output): ConvDropoutLayerNorm(
    (conv1d): Conv1d(3072, 768, kernel_size=(1,), stride=(1,), groups=4)
    (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1)
  )
)
(3): SqueezeBertModule(
  (attention): SqueezeBertSelfAttention(
    (query): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (key): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (value): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (dropout): Dropout(p=0.1)
    (softmax): Softmax()
    (matmul_qk): MatMulWrapper()
    (matmul_qkv): MatMulWrapper()
  )
  (post_attention): ConvDropoutLayerNorm(
    (conv1d): Conv1d(768, 768, kernel_size=(1,), stride=(1,))
```

```
      (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1)
    )
    (intermediate): ConvActivation(
      (conv1d): Conv1d(768, 3072, kernel_size=(1,), stride=(1,), groups=4)
    )
    (output): ConvDropoutLayerNorm(
      (conv1d): Conv1d(3072, 768, kernel_size=(1,), stride=(1,), groups=4)
      (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1)
    )
  )
  (4): SqueezeBertModule(
    (attention): SqueezeBertSelfAttention(
      (query): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
      (key): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
      (value): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
      (dropout): Dropout(p=0.1)
      (softmax): Softmax()
      (matmul_qk): MatMulWrapper()
      (matmul_qkv): MatMulWrapper()
    )
    (post_attention): ConvDropoutLayerNorm(
      (conv1d): Conv1d(768, 768, kernel_size=(1,), stride=(1,))
      (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1)
    )
    (intermediate): ConvActivation(
      (conv1d): Conv1d(768, 3072, kernel_size=(1,), stride=(1,), groups=4)
    )
    (output): ConvDropoutLayerNorm(
      (conv1d): Conv1d(3072, 768, kernel_size=(1,), stride=(1,), groups=4)
      (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1)
    )
  )
  (5): SqueezeBertModule(
    (attention): SqueezeBertSelfAttention(
      (query): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
      (key): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
      (value): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
      (dropout): Dropout(p=0.1)
      (softmax): Softmax()
      (matmul_qk): MatMulWrapper()
      (matmul_qkv): MatMulWrapper()
    )
    (post_attention): ConvDropoutLayerNorm(
      (conv1d): Conv1d(768, 768, kernel_size=(1,), stride=(1,))
      (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1)
    )
    (intermediate): ConvActivation(
      (conv1d): Conv1d(768, 3072, kernel_size=(1,), stride=(1,), groups=4)
    )
    (output): ConvDropoutLayerNorm(
      (conv1d): Conv1d(3072, 768, kernel_size=(1,), stride=(1,), groups=4)
      (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1)
    )
  )
  (6): SqueezeBertModule(
    (attention): SqueezeBertSelfAttention(
      (query): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
      (key): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
      (value): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
      (dropout): Dropout(p=0.1)
      (softmax): Softmax()
      (matmul_qk): MatMulWrapper()
      (matmul_qkv): MatMulWrapper()
    )
    (post_attention): ConvDropoutLayerNorm(
      (conv1d): Conv1d(768, 768, kernel_size=(1,), stride=(1,))
      (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1)
    )
    (intermediate): ConvActivation(
      (conv1d): Conv1d(768, 3072, kernel_size=(1,), stride=(1,), groups=4)
    )
    (output): ConvDropoutLayerNorm(
      (conv1d): Conv1d(3072, 768, kernel_size=(1,), stride=(1,), groups=4)
      (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1)
    )
  )
```

```
(7): SqueezeBertModule(
  (attention): SqueezeBertSelfAttention(
    (query): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (key): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (value): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (dropout): Dropout(p=0.1)
    (softmax): Softmax()
    (matmul_qk): MatMulWrapper()
    (matmul_qkv): MatMulWrapper()
  )
  (post_attention): ConvDropoutLayerNorm(
    (conv1d): Conv1d(768, 768, kernel_size=(1,), stride=(1,))
    (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1)
  )
  (intermediate): ConvActivation(
    (conv1d): Conv1d(768, 3072, kernel_size=(1,), stride=(1,), groups=4)
  )
  (output): ConvDropoutLayerNorm(
    (conv1d): Conv1d(3072, 768, kernel_size=(1,), stride=(1,), groups=4)
    (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1)
  )
)
(8): SqueezeBertModule(
  (attention): SqueezeBertSelfAttention(
    (query): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (key): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (value): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (dropout): Dropout(p=0.1)
    (softmax): Softmax()
    (matmul_qk): MatMulWrapper()
    (matmul_qkv): MatMulWrapper()
  )
  (post_attention): ConvDropoutLayerNorm(
    (conv1d): Conv1d(768, 768, kernel_size=(1,), stride=(1,))
    (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1)
  )
  (intermediate): ConvActivation(
    (conv1d): Conv1d(768, 3072, kernel_size=(1,), stride=(1,), groups=4)
  )
  (output): ConvDropoutLayerNorm(
    (conv1d): Conv1d(3072, 768, kernel_size=(1,), stride=(1,), groups=4)
    (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1)
  )
)
(9): SqueezeBertModule(
  (attention): SqueezeBertSelfAttention(
    (query): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (key): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (value): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (dropout): Dropout(p=0.1)
    (softmax): Softmax()
    (matmul_qk): MatMulWrapper()
    (matmul_qkv): MatMulWrapper()
  )
  (post_attention): ConvDropoutLayerNorm(
    (conv1d): Conv1d(768, 768, kernel_size=(1,), stride=(1,))
    (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1)
  )
  (intermediate): ConvActivation(
    (conv1d): Conv1d(768, 3072, kernel_size=(1,), stride=(1,), groups=4)
  )
  (output): ConvDropoutLayerNorm(
    (conv1d): Conv1d(3072, 768, kernel_size=(1,), stride=(1,), groups=4)
    (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1)
  )
)
(10): SqueezeBertModule(
  (attention): SqueezeBertSelfAttention(
    (query): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (key): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (value): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
    (dropout): Dropout(p=0.1)
    (softmax): Softmax()
    (matmul_qk): MatMulWrapper()
    (matmul_qkv): MatMulWrapper()
  )
  (post_attention): ConvDropoutLayerNorm(
    (conv1d): Conv1d(768, 768, kernel_size=(1,), stride=(1,))
```

```
        (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1)
      )
      (intermediate): ConvActivation(
        (conv1d): Conv1d(768, 3072, kernel_size=(1,), stride=(1,), groups=4)
      )
      (output): ConvDropoutLayerNorm(
        (conv1d): Conv1d(3072, 768, kernel_size=(1,), stride=(1,), groups=4)
        (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1)
      )
    )
    (11): SqueezeBertModule(
      (attention): SqueezeBertSelfAttention(
        (query): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
        (key): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
        (value): Conv1d(768, 768, kernel_size=(1,), stride=(1,), groups=4)
        (dropout): Dropout(p=0.1)
        (softmax): Softmax()
        (matmul_qk): MatMulWrapper()
        (matmul_qkv): MatMulWrapper()
      )
      (post_attention): ConvDropoutLayerNorm(
        (conv1d): Conv1d(768, 768, kernel_size=(1,), stride=(1,))
        (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1)
      )
      (intermediate): ConvActivation(
        (conv1d): Conv1d(768, 3072, kernel_size=(1,), stride=(1,), groups=4)
      )
      (output): ConvDropoutLayerNorm(
        (conv1d): Conv1d(3072, 768, kernel_size=(1,), stride=(1,), groups=4)
        (layernorm): SqueezeBertLayerNorm(torch.Size([768]), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1)
      )
    )
  )
)
(pooler): SqueezeBertPooler(
  (dense): Linear(in_features=768, out_features=768, bias=True)
  (activation): Tanh()
)
```

```
 )
)
(ext_layer): ExtTransformerEncoder(
 (pos_emb): PositionalEncoding(
  (dropout): Dropout(p=0.2)
 )
 (transformer_inter): ModuleList(
  (0): TransformerEncoderLayer(
   (self_attn): MultiHeadedAttention(
    (linear_keys): Linear(in_features=768, out_features=768, bias=True)
    (linear_values): Linear(in_features=768, out_features=768, bias=True)
    (linear_query): Linear(in_features=768, out_features=768, bias=True)
    (softmax): Softmax()
    (dropout): Dropout(p=0.2)
    (final_linear): Linear(in_features=768, out_features=768, bias=True)
   )
   (feed_forward): PositionwiseFeedForward(
    (w_1): Linear(in_features=768, out_features=2048, bias=True)
    (w_2): Linear(in_features=2048, out_features=768, bias=True)
    (layer_norm): LayerNorm(torch.Size([768]), eps=1e-06, elementwise_affine=True)
    (dropout_1): Dropout(p=0.2)
    (dropout_2): Dropout(p=0.2)
   )
   (layer_norm): LayerNorm(torch.Size([768]), eps=1e-06, elementwise_affine=True)
   (dropout): Dropout(p=0.2)
  )
  (1): TransformerEncoderLayer(
   (self_attn): MultiHeadedAttention(
    (linear_keys): Linear(in_features=768, out_features=768, bias=True)
    (linear_values): Linear(in_features=768, out_features=768, bias=True)
    (linear_query): Linear(in_features=768, out_features=768, bias=True)
    (softmax): Softmax()
    (dropout): Dropout(p=0.2)
    (final_linear): Linear(in_features=768, out_features=768, bias=True)
   )
   (feed_forward): PositionwiseFeedForward(
    (w_1): Linear(in_features=768, out_features=2048, bias=True)
    (w_2): Linear(in_features=2048, out_features=768, bias=True)
    (layer_norm): LayerNorm(torch.Size([768]), eps=1e-06, elementwise_affine=True)
    (dropout_1): Dropout(p=0.2)
    (dropout_2): Dropout(p=0.2)
   )
```

```
        (layer_norm): LayerNorm(torch.Size([768]), eps=1e-06, elementwise_affine=True)
        (dropout): Dropout(p=0.2)
      )
    )
    (dropout): Dropout(p=0.2)
    (layer_norm): LayerNorm(torch.Size([768]), eps=1e-06, elementwise_affine=True)
    (wo): Linear(in_features=768, out_features=1, bias=True)
    (sigmoid): Sigmoid()
  )
)
```