

# **An In-Depth Exploration of Abstractive Text Summarization Utilizing Deep Learning**

**A MINI PROJECT REPORT**

**18CSE359T – Natural Language Processing**

*Submitted by*

**OM TIWARI(RA2011026010341)  
GEETHA SHASHANK(RA2011026010345)  
MEET KACHHARA(RA2011026010230)  
EKTA KHANDELWAL(RA2011026010307)**

*Under the guidance of*  
**Dr.Dhilsath Fathima M**

Assistant Professor, Department of Computer Science and Engineering

*in partial fulfillment for the award of the degree of*

**BACHELOR OF TECHNOLOGY**

in

**COMPUTER SCIENCE & ENGINEERING**

of

**FACULTY OF ENGINEERING AND TECHNOLOGY**



S.R.M. Nagar, Kattankulathur, Chengalpattu District

**NOVEMBER 2023**

# **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

(Under Section 3 of UGC Act, 1956)

## **BONAFIDE CERTIFICATE**

Certified that Mini project report titled **“An In-Depth Exploration of Abstractive Text Summarization Utilizing Deep Learning”** is the bona fide work of **“OM TIWARI (RA2011026010341), GEETHA SASHANK(RA2011026010345), MEET KACHHARA (RA2011026010230), EKTA KHANDELWAL(RA2011026010307)”** who carried out the minor project under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

**SIGNATURE**

Dr.Dhilsath Fathima M  
GUIDE  
Assistant Professor  
Department of Computational Intelligence

**SIGNATURE**

Dr. R.Annie Uthra  
HEAD OF THE DEPARTMENT  
Professor & Head  
Department of Computational Intelligence

# SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

## BONAFIDE CERTIFICATE

ified that Mini project report titled "An In-Depth Exploration of Abstractive Text  
rization Utilizing Deep Learning" is the bona fide work of "OM TIWARI  
6010341), GEETHA SASHANK(RA2011026010345), MEET KACHHARA (RA2011026010230), EKTA  
LWAL(RA2011026010307)" who carried out the minor project under my supervision.  
d further, that to the best of my knowledge, the work reported herein does not form any  
object report or dissertation on the basis of which a degree or award was conferred on an  
occasion on this or any other candidate.

SIGNATURE

*Dr. Dhilsath Fathima M*  
11/11/23

Dr.Dhilsath Fathima M

GUIDE

Assistant Professor

Department of Computational Intelligence

SIGNATURE

*Dr. R. Annie Uthra*

Dr. R. Annie Uthra

HEAD OF THE DEPARTMENT

Professor & Head

Department of Computational Intelligence

**ABSTRACT**

The rapid expansion of the Internet has led to an exponential increase in the volume of textual data available online. This vast amount of web-based textual information poses significant challenges for downstream tasks such as document management, text classification, and information retrieval. Automatic text summarization (ATS) has emerged as a critical solution to cope with this information overload. ATS focuses on distilling the essence of the source text and generating concise, coherent, and readable summaries automatically. In recent years, deep learning (DL)-based abstractive summarization models have gained prominence, aiming to strike a balance between informativeness and fluency. Currently, nearly all state-of-the-art (SOTA) ATS models are rooted in deep learning architectures. Despite the prevalence of these models, there exists a notable gap in the literature—a comprehensive survey of DL-based abstractive text summarization is conspicuously missing. To address this deficiency and offer a valuable resource to researchers, this paper presents an exhaustive exploration of DL-based abstractive summarization. The survey commences with an introduction to abstractive summarization and deep learning, providing essential background information. Subsequently, we delve into various prominent frameworks employed for abstractive summarization. We also furnish a comparative analysis of popular datasets that are widely utilized for training, validation, and testing in this domain. Additionally, we conduct a thorough assessment of the performance of several representative abstractive summarization systems on these common datasets. This evaluation serves to offer insights into the capabilities and limitations of existing models. As we progress, we outline some of the outstanding challenges in the field of abstractive summarization and delineate potential directions for future research. By identifying and discussing these research gaps, we aim to inspire innovative approaches and solutions in the realm of abstractive summarization. Our objective is to equip researchers with a holistic understanding of DL-based abstractive text summarization and inspire them to contribute to this evolving field. We anticipate that this comprehensive survey will not only facilitate a deeper appreciation of the subject but also encourage novel insights and advancements in the domain of ATS.

# TABLE OF CONTENTS

<b>ABSTRACT</b>	<b>iii</b>
<b>TABLE OF CONTENTS</b>	<b>iv</b>
<b>LIST OF FIGURES</b>	<b>v</b>
<b>ABBREVIATIONS</b>	<b>vi</b>
<b>1 INTRODUCTION</b>	<b>7</b>
<b>2 LITERATURE SURVEY</b>	<b>12</b>
<b>3 SYSTEM ARCHITECTURE AND DESIGN</b>	<b>13</b>
3.1 Models	13
3.2 RNN	14
3.3 CNN	15
3.4 LSTM and GRU	15
<b>4 METHODOLOGY</b>	<b>16</b>
4.1 Seq2Seq Framework	16
4.2 Encoder-Decoder Systems with Basic Attention Mechanism	16
4.3 Hierarchical Encoder-Decoder Models	18
<b>5 CODING AND TESTING</b>	<b>49</b>
<b>6 RESULTS</b>	<b>50</b>
<b>7 CONCLUSION AND FUTURE ENHANCEMENT</b>	<b>51</b>
7.1 Conclusion	
7.2 Future Enhancement	
<b>REFERENCES</b>	<b>52</b>

**LIST OF FIGURES**

3.1	Encoder-Decoder framework	16
3.2	Encoder-decoder framework (attention mechanisms)	17
5.1	Word Cloud	25
5.2	Accuracy	49
6.1	Loss function	49

## ABBREVIATIONS

<b>RCNN</b>	Region-based Convolutional Neural Network
<b>CNN</b>	Convolution Neural Network
<b>LSTM</b>	Long Short-Term Memory
<b>DL</b>	Deep Learning
<b>NLP</b>	Natural Language processing
<b>DDP</b>	Differential Dynamic Programming
<b>ATS</b>	Automatic text summarization

# CHAPTER 1

## INTRODUCTION

In the digital age, the vast landscape of online resources, encompassing web pages, blogs, news articles, user-generated content, and social media platforms, has given rise to an unprecedented proliferation of textual data. This exponential growth presents a substantial challenge for various downstream tasks, including document management, text classification, and information retrieval. Users are constantly confronted with the formidable task of sifting through copious volumes of text to extract pertinent information, a process that significantly hampers their efficiency [1–11]. Therefore, the rapid and precise location, summarization, and compression of essential information from these textual resources have become urgent and fundamental imperatives. Traditional manual summarization methods involve the laborious process of manually reviewing and summarizing text, an approach that is not only costly but easily overwhelmed by the sheer volume of data. In response to this dilemma, automatic text summarization (ATS) has emerged as an effective solution [12–21].

ATS aims to automatically generate concise and coherent summaries that capture the essence of the input text. It has become increasingly pivotal in the quest for expeditious, reliable, and efficient access to required information. Owing to the intricate nature of input text, ATS has evolved into one of the most formidable challenges within the domain of natural language processing (NLP) [22–34]. Its origins can be traced back to 1958 when Luhn [35] embarked on early ATS studies, proposing automated extraction of summaries from magazine articles and technical papers. Subsequently, in 1995, Maybury [36] devised a system capable of extracting key information from event databases, defining a high-quality summary as the most crucial content distilled from the input document. In 2002, Radev et al. [37] introduced the concept of summarization as a composite of sentences generated from one or multiple input documents, emphasizing that the length of the summary should not exceed half of the input or even less. These foundational descriptions encapsulate the fundamental characteristics of ATS, emphasizing the need for summaries to encapsulate the core content of the input text while maintaining conciseness. Broadly, there are two predominant summarization systems classified by their approach to summary generation: extractive summarization [38–41] and abstractive summarization (ABS) [42–53]. Extractive systems directly select sentences or phrases from the source document to form a summary, employing techniques such as graph-based (e.g., LexRank [54]), centrality-based (e.g., Centroid [55]), and corpus-based (e.g., TsSum [56]) methods. In contrast, abstractive systems require a comprehensive understanding of the text's semantics and employ natural language generation (NLG) algorithms to create more concise summaries through paraphrasing, synonymous substitution, sentence compression, and other linguistic processes. Consequently, abstractive summarization closely resembles the way humans generate



summaries manually [57]. However, the progress of abstractive summarization has been relatively slow due to the limitations of conventional methods concerning textual representation, comprehension, and generation capabilities, with extractive summarization often outperforming abstractive methods [58].

Recent advancements in deep learning (DL) have ushered in a new era of effective and promising methodologies, leading to state-of-the-art (SOTA) performance across a wide array of tasks, including image processing, computer vision, natural language generation (NLG), and natural language processing (NLP) [59–66]. In 2015, Rush et al. [67] laid the foundation for DL-based abstractive summarization, introducing an ABS model built upon the encoder-decoder architecture. Subsequently, a slew of enhanced ABS models have emerged, all rooted in the encoder-decoder paradigm. To date, the research community's enthusiasm for DL-based abstractive summarization remains fervent, with many outstanding methodologies continually pushing the boundaries of performance. With an expanding cohort of researchers dedicated to ABS research, the time is ripe for an overarching perspective to provide them with comprehensive insights into the accomplishments and challenges within this domain. This paper endeavors to fulfill this need. Our focus is on DL-based abstractive summarization tasks and their developmental trajectory. We present an overview of prevalent foundational frameworks and improved methodologies. We offer a critical analysis of the existing models, objectively enumerating their strengths and limitations. Furthermore, we conduct a comparative evaluation of these models using popular evaluation metrics on large-scale public datasets. Finally, we delineate unresolved challenges in the field of abstractive summarization and outline future avenues for research. In particular, our approach extends to four distinct dimensions: (1) Methodology categorization to classify recent models; (2) Introducing a novel model type that addresses factual inaccuracies and conducting a comprehensive analysis; (3) Summarizing the ROUGE scores of all SOTA models in the past five years, offering a visual representation of DL-based summarization's evolution; (4) Delving into potential future research foci from an application-centric perspective. In essence, the explosive growth of textual data in the digital age has given rise to an ever-increasing need for ATS. Deep learning methods have revolutionized abstractive summarization, and this survey seeks to provide a comprehensive overview of this field, from its historical roots to its most recent advances, culminating in a glimpse of what the future holds. Our work makes significant contributions in the following ways:

(i) We offer a structured examination of DL-based abstractive summarization (ABS) methodologies and provide intricate insights into various well-regarded frameworks operating within the encoder-decoder architecture.

(ii) We categorize DL-based ABS models, expound on the characteristics of each class, and conduct a thorough analysis of their respective merits and drawbacks.

(iii) We furnish an all-encompassing overview of commonly employed datasets and evaluation metrics in the ABS domain. Furthermore, we present a detailed performance assessment of diverse models on extensive datasets, offering valuable guidance to researchers in selecting an appropriate framework and model tailored to their specific requirements.

(iv) We engage in a discourse surrounding several promising research directions and offer fresh perspectives and inspirations for future investigations and the practical applications.

## **CHAPTER 2**

### **LITERATURE SURVEY**

In recent years, vast quantity of documents is available on the web, and the number is growing exponentially over time, which creates the problem of information overload. Hence, reading only the most important information while skipping the minor parts in a short time will save the time and effort for readers to read various “summarized” documents for a given field. For example, in the field of scientific articles, a large number of research papers are published daily, for example, Pubmed publishes 1.5 papers per minute<sup>1</sup>. Moreover, hot research areas, such as COVID-19 research, require speeding up the process of summarizing details and findings to bridge the gap between researchers.

Currently, ATS is one of the most popular research areas in NLP. ATS can be categorized into two main types. The first being extractive, i.e., highlighting, and the second being abstractive, i.e., paraphrasing. Clearly, abstractive summarization is more challenging as it deals with higher-level aspects such as semantic representation, surface realization, and content organization [25]. Moreover, abstractive ATS requires an automated understanding of the input text beyond the meanings of words and sentences, which is a challenging task. Generally, the main requirements for abstractive summaries include fluency, saliency, coherency, information correctness, and novelty [3].

Compared to extractive summarization, the abstractive type of ATS is distinguished by its readability, consistency, and conciseness. However, it is more complex, requires higher computational costs, and has the tendency to produce trivial summaries with factual mistakes, redundant information, and major information loss. However, recent advances

in abstractive ATS research have shown impressive results that overcome these shortcomings, particularly with the use of modern approaches such as PTLMs and the knowledge-enhanced mechanism. Additionally, a multi-document summarization experiment made by Genest et al., [61] to compare human-written summaries showed that the performance of pure extractive summaries is very low compared to abstractive summaries.

Recently, there has been an increasing interest in utilizing both the extractive and abstractive types of ATS, by combining them together to enhance the quality of the resulting final abstractive summary. This mixed-method is applied first using DL-based techniques; pointer-generator [14, 15, 16, 17, 18] and knowledge-enhanced [49,62,63], which are discussed in Section 3.1.3 and Section 3.1.5, respectively and then by RL-based fusion techniques [19, 20, 21, 22, 23,64], discussed in Section 4.2. In this article, based on the final result generated, we use the term “abstractive ATS models” to refer to both mixed models and pure abstractive ATS models as the end result of both models is the same, which is an abstractive summary with new phrases.

In general, humans tend to generate abstractive forms using the mixed approach when writing their summaries. They first extract the most salient parts and kinds of information from the text and then paraphrase them into a readable and coherent form. There is no doubt that human-generated summaries are more reliable and efficient because human cognition and experience play a primary role in the summarization process. Also, human summarizers can efficiently interpret a document, prioritize important sentences, and create diverse coherent paraphrased versions of the summary [65]. However, manual text summarization is time and effort consuming.

These concerns sparked interest in the research of abstractive ATS. That is why massive amounts of research have been conducted in the abstractive text summarization field, covering all its aspects. However, abstractive ATS research still requires more effort in attempting new strategies to reach the level of human-generated summaries.

There are two main types of ATS which are extractive and abstractive. In the extractive type of ATS, the summary concatenates the original document's most important sentences without any modification. On the other hand, abstractive summaries are completely new sentences with the same meaning as the ideas of the original document. Technically, the extractive type can be loosely defined as binary classification, i.e., whether the source article's token is to be included in the generated summary. On the other hand, abstractive summarization requires more sophisticated approaches to generate a novel text. Extractive method is considered to be more simplistic, thus, its research is being explored more widely in contrast to the abstractive ATS research.

Other categories can be categorized based on different perspectives, including length, input, output, purpose, language, and more. Table 1 summarizes the different categories and types of ATS.

Some types of ATS can be combined, while others are incompatible. For example, several researchers nowadays demonstrate that the hybrid model of extractive and abstractive types of ATS achieves better results than the stand-alone abstractive ATS model using sequence-to-sequence techniques 14, 15, 16, 17, 18, and RL approaches 19, 20, 21, 22, 23. Another example, Wang et al., [24] combined compression, query-based, and multi-document summarizations into one summarizer. On the other hand, search engines, which are query-based summarizations, cannot summarize multi-documents even with advanced information retrieval tools.

Generally, most researchers work on the English language, news context, and text output types of ATS. This study focuses on research works related to abstractive, short- and long- document, single-document, textual, generic, and English types of ATS. From Table 1, some of these categories were collected by [2,25], and [26] survey papers.

## CHAPTER 3

### SYSTEM ARCHITECTURE AND DESIGN

#### 3.1.MODELS

Sequence data are time-based data where the arrangement of its components, i.e., timesteps, is critical, including number sequences, text sequences, video frame sequences, and audio sequences. For example, the order of the words that compose a sentence plays an essential role in understanding the whole sentence. RNNs [69] are primarily designed in a sequence nature, so they are the most appropriate DL architectures for encoding and processing sequence data such as text. RNNs can handle sequence dependencies between input tokens (characters, words, sentences, etc.) as well as the variable-length input for unstructured data such as text. Moreover, RNNs are able to generate feedback to prior layers to maintain memory inputs. Therefore, the output of some layers can be fed back into the inputs of previous layers, giving RNNs the ability to analyse sequential data. Unidirectional RNNs can only capture the context based on the token's history, and they do not consider all the context required from the future aspect. This may lead to misleading predictions and, as a result, incorrect information generation. In human perception, to predict an unknown word in a sentence, the context of both sides is taken into consideration to ensure the correct prediction. Therefore, using bidirectional RNN allows each hidden state to be aware of the contextual information from both directions, i.e., past and future contexts, in order to improve the results. Al-Sabahi et al., [70] proposed a bidirectional encoder-decoder model that impressively improves the results of generated summaries and overcomes the problem of generating wrong information.

#### 3.2 RNN

However, vanilla RNNs suffer from vanishing and exploding gradient problems during training, and therefore are unable to model long-term dependencies when the sequence length increases. Some variants of RNN, called canonical architectures, i.e., Long Short-Term Memory (LSTM) [71] and Gated Recurrent Units (GRU) [67], have been found to model long-term dependencies more accurately and overcome the

gradient-vanishing problem [72] using gating mechanisms. Note that GRU is a simplification of LSTM with simpler resources, less complexity and operating cost, and faster training and execution. However, the exploding gradient problem could be solved via other techniques such as gradient norm clipping strategy [73]. Nevertheless, LSTM and GRU still struggle to handle very long-term memory modeling in very long documents, and thus cannot copy or recall information from a long-distance past. Recently, the Rotational Unit of Memory (RUM) [74] has been proposed as a representation unit for RNNs that unifies the properties of unitary learning and associative memory to handle memory copying and memory recall tasks better than LSTMs and GRUs. RNN based on RUM module models can better handle long-term dependencies enhancing ROUGE results when replacing LSTM and GRU. In particular, models that incorporate RUM units lead to larger gradients during training resulting in more stable training and better convergence [75]. Most researchers choose to use RNNs for NLP tasks, especially for language modelling tasks. RNN variants have gained popularity in modelling sequence data due to their sequential nature and their ability to capture unlimited and unstructured contexts. However, training with RNN is limited to parallel processing on sequence elements due to their temporal dependencies between hidden states [76]. Furthermore, they must also preserve the hidden state of the whole past [77] because of their sequential dependence nature [78], and they are still restricted to handle long-term dependencies for very long documents due to their linear path length between the output and any of the input tokens [65]. Therefore, the feedforward architecture, Convolutional Neural Networks CNNs [79], especially hierarchical CNNs, can be used instead as fundamental networks to model sequential data [28,76, 77, 78,80].

### 3.3 CNN

CNNs could efficiently address previous RNN's issues by computing each element in the sequence in parallel during training and evaluation, which, as a result, will lead to improved efficiency and performance of the model [80]. Other advantages of using CNNs with sequence-to-sequence models include their ease of optimizing, their linear computational complexity, and more efficient propagation of gradient signals [78]. Dauphin et al. [81] apply gated CNNs to language modelling and prove that CNNs allow parallelization over sequential data and achieve competitive results with robust

gated recurrent LSTM models. Moreover, Zhang et al., [77] apply a hierarchical CNN framework for abstractive ATS, outperforming most RNN models. For ATS, both RNN and CNN architectures are used to create abstractive and extractive summaries. For example, IBM has developed an abstractive summarization model based on RNN [16]. Also, Facebook has developed a similar model based on CNN [28]. More recently, significant improvements in the quality of results and ROUGE scores have been obtained using the Transformer architecture [4]. This fully attention-based architecture has been developed based on attention mechanisms only without any use of RNN or CNN nodes. Various models can utilize Transformer, such as PTLMs, as an encoder and/or decoder. Essentially, Transformer is used to model similarities between input tokens regardless of their positions in parallel by attending each token in the input sequence independently using the self-attention mechanism. Therefore, Transformer efficiently solved the problems of RNN, i.e., sequential nature and long-term dependencies. As a result, models that are based on the attention-based architecture, Transformer, outperformed RNN- and CNN-based sequence-to-sequence models and showed SotA results on various NLP tasks in less training time, including abstractive ATS as illustrated. However, more efficient Transformers and attentions have recently been introduced to overcome the Transformer limitations of quadratic memory complexity, the large number of operations required, and the fixed-length context prerequisite. These improvements.

### 3.4 LSTM and GRU

Moreover, combining the advantages of attention and recurrence has recently emerged as a research trend. Many kinds of work have been done in NLP research to modify the architecture of internal Transformers and combine it with recurrent units, i.e., GRUs and LSTMs. This process showed significant improvements in terms of speed and results in various NLP tasks [82, 83, 84, 85, 86].

The popularity of DL nowadays mainly arises after the availability of numerous annotated datasets and the availability of computational resources that facilitate the use of parallel processing through modern Single Instruction-Multiple Data (SIMD) hardware accelerators such as GPUs and TPUs. GPUs and TPUs make the processing faster, cheaper, and more powerful. Despite the significant improvements resulting from using

deep sequence-to-sequence models and their various mechanisms in NLP tasks, RNN-based models are still limited to various weaknesses. For example, they cannot work in parallel due to their sequential design nature. Thus, they can't leverage the full power of GPUs and TPUs. Moreover, they cannot efficiently handle long-range dependencies of long document inputs, resulting in low-quality summaries. Other problems include low-novelty, exposure-bias problem, loss/evaluation mismatch, and lack of generalization. Therefore, integrating RL and TL approaches can efficiently solve these problems. RL rewarding can help improve abstractive ATS results in its various aspects, such as ROUGE scores, quality, factual consistency, novelty, readability, coherency, syntax, non-redundancy, sentence ordering, conciseness, information diversity, information coverage, saliency, and entailment [87]. Moreover, as discussed in Section 5, utilizing Transformers' parallelization allows models to take advantage of the full power of GPUs and TPUs and increase training speed even with massive datasets. PTLMs pre-train the Transformers on huge corpora and transfer their acquired knowledge to downstream tasks such as abstractive ATS. As a result, the universal and rich semantic and contextual features of word embeddings acquired by PTLMs prove that they can assist in enhancing the quality of the resulting summaries. This widespread success of Transformers and PTLMs for various NLP tasks, including abstractive ATS, makes them the backbone of recent NLP research.

## 2.2. Deep Neural Network

Deep neural networks (DNNs) are the foundation of deep learning, which use sophisticated mathematical methods to train various models. It contains many hidden layers, so it is sometimes called a multi-layer perceptron (MLP). In this section, we introduce several DNNs commonly used in ABS, including recurrent neural networks (RNN), convolutional neural network (CNN), and graph neural network (GNN). The proposal of RNN is based on an intuitive understanding that "human's cognition is based on experience and memory." In RNN, there is a sequential relationship within the sequence, and adjacent items depend on each other. The network predicts the output of the next time step by combining the characteristics of the input at the previous and the current timestep. Specifically, the hidden layer nodes of RNN are connected to each other. The hidden layer input is composed of the output of the input layer and the previous hidden layer.



## CHAPTER 4

### METHODOLOGY

In this section, we review and summarize the development of ABS from the perspective of methodology.

#### 4.1. Seq2Seq Framework

The Seq2seq (sequence-to-sequence) framework, also known as the encoder-decoder framework, is widely regarded as the most efficient method in converting text from one form to another, such as speech recognition, question answering system, machine translation, etc. These models employ an encoder to identify, understand, and parse the input sequence, and use the high-dimensional dense feature vector to characterize it. Then, on the decoder side, the feature vectors of the input items are used to generate the output items gradually. Figure 5 shows the basic encoder-decoder framework. The encoder-decoder framework is the most basic and core framework of DL-based ABS models. And, the encoder and decoder are constructed using various neural networks. A large number of research results are put forward based on encoder-decoder architecture [22–24], which makes the performance of the ABS models continuously improved.

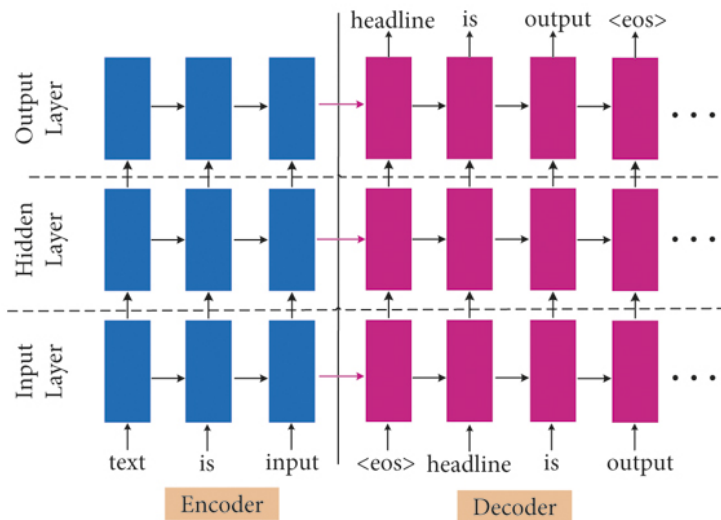


Fig. The basic encoder-decoder framework. It consists of input layer, hidden layer, and output layer.

## 4.2. Encoder-Decoder Systems with Basic Attention Mechanism

In 2015, Rush et al. [67] applied the encoder-decoder framework to the ABS for the first time. They proposed a novel ABS model with an attention mechanism. The model is mainly composed of the feed-forward neural language model (FFNLM), which is a parameterized neural network. The most significant advantage of their system is the use of a more powerful attention-based encoder (vs. Bag-of-Words encoder) and a beam search strategy [72] (vs. greedy decoding) to generate summaries.

After that, Chopra et al. [73] further proposed a convolutional RNN model for ABS, which is an extension of the method proposed by Rush et al. [67]. The encoder of their model adopts a convolutional attention mechanism to ensure that the decoder aligns with the corresponding input token at each decoding time step, thus providing an adjustment function for the generation process. In addition, they also provided two optional networks for the decoder: Vanilla RNN and LSTM. The encoder-decoder framework with an attention mechanism [73]. The attention-based context vector is calculated as in equations (3)–(5) [73]:

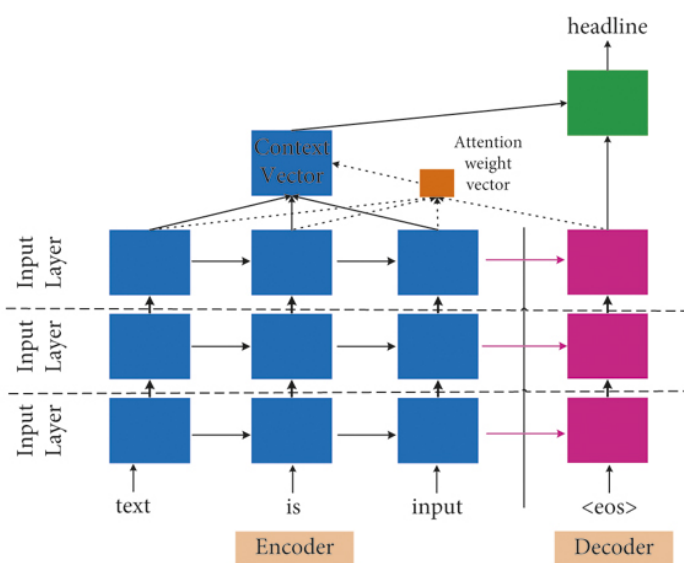


Fig. The basic encoder-decoder framework with attention mechanisms. The attention mechanism enables the decoder to interact with the input during the decoding process.

Lopyrev et al. [74] tested two different attention mechanisms in the news headlines generation task. The first one is the same as the dot mechanism in Figure 5, and they called it complex attention. The second one is a slight variation of the dot mechanism consisting of some neurons used to calculate the attention weights, which has specific advantages when further exploring the functions of the network, and they called it simple attention. Their experiments showed that the simple attention mechanism performed better. Chen et al. [75] utilized the distraction-based Bi-GRU to model input document. In order to better model the overall document representation, they focused on specific regions and contents of the input text, while also distracting them from traversing between different contents of the input text. Their work is the early application of the coverage mechanism in ABS. However, because RNN is difficult to control during the generation process, the basic encoder-decoder architecture still has some critical problems in ABS, such as generating out-of-vocabulary (OOV) words, modeling keywords, and capturing the hierarchical structure of words to sentences. To alleviate these problems, Nallapati et al. [76] further extended the basic encoder-decoder model. They constructed a feature-rich encoder, which uses an embedding vector for the Part-of-Speech (POS), named Entity Recognition (NER) tags, and discretized TF and IDF values, respectively. Then, these values are connected with word-based embedding values as the encoder input. The feature-rich-encoder can capture the key concepts and entities in the input document. They also employed a switching generator-pointer to model rare/unseen words in the input document, which alleviates the problem of generating OOV words. Moreover, they also introduced hierarchical attention to jointly model key sentences and keywords in the key sentences.

Furthermore, when processing longer documents (usually more than 1000 tokens), neural network-based models often generate repeated words and phrases, and even inconsistent phrases. To alleviate these problems, Paulus et al. [77] adopted the intra-attention method, which can pay attention to the specific area of input tokens and continuously generate output separately. At each decoding step, in addition to the decoder's hidden state and the previously generated tokens, their model also employs an intra-temporal attention function to pay attention to the specific area of input text. Thus, the intra-temporal

attention can prevent the model from repeatedly paying attention to the same part in the original document at different decoding timesteps. To solve the problem of generating repeated phrases based on encoder hidden states, they further proposed to utilize intra-decoder attention to incorporate more information about the previously generated tokens into the decoder. At the current decoding time step, considering the tokens that have been generated allows the encoder-decoder model to make a more holistic decision, which can effectively avoid generating duplicate tokens, even if these tokens are generated many steps away.

#### 4.3. Hierarchical Encoder-Decoder Models

When the input is a lengthy document, the basic single-layer encoder-decoder architecture cannot fully capture the relationship between the contexts when encoding the document, which leads to the problem of long-distance dependence. Researchers found that long documents naturally have a hierarchical structure, that is, documents are composed of multiple long sentences (sentence level), and long sentences are composed of multiple words (word level). Inspired by this, researchers constructed a hierarchical encoder-decoder architecture. The hierarchical encoder-decoder architecture can significantly reduce long dependency problems. The basic framework of the hierarchical encoder-decoder ABS. Hierarchical neural models have shown strong performance in document-based language models (LM) [78] and some document classification [79] tasks. In 2015, Li et al. [80] proposed a basic hierarchical ABS model, and Jadhav and Rajan [81] further extended their model. And the summaries generated by their method are significantly better than similar methods in terms of informativity and readability. Inspired by the graph-based NLP models, Tan et al. [82] proposed a novel graph-based attention mechanism in the hierarchical encoder-decoder framework. They employed a word encoder to encode words, used a sentence encoder to encode short sentences, and utilized the hidden state of the sentences to construct a hidden state graph. The hierarchical attention value of the sentence is calculated from a hidden state graph.

Although the above hierarchical encoder-decoder model is designed based on the

sentence-word hierarchy, it fails to capture the global structural characteristics of the document. In 2018, Li et al. [83] used the structural information of multi-sentence summaries and documents to enhance the performance of ABS models. In order to mine the information compression and information coverage properties, they proposed to model structural-compression and structural-coverage regularization during summary generation. They utilized sentence-level attention distributions to calculate the score of the structural-compression.

#### 4.4. CNN-Based Encoder-Decoder Models

Unlike RNN that directly processes time-series data, CNN uses convolution kernels to extract features from data objects, which are often used in image-related tasks [85]. But after the text is represented by a distributed vector, each token is a matrix in the vector space. Then CNN can be used to perform convolution operations in text-related tasks [86]. In 2016, Facebook AI Research (FAIR) used CNN to build an encoder under the encoder-decoder architecture for the first time, and achieved SOTA results in machine translation tasks [87]. In 2017, Gehring et al. [88] proposed a model ConvS2S and its encoder and decoder both use CNN, which is the most representative ABS model based entirely on CNN. The overall architecture of the model is shown in Figure 8 [88]. In their model, in addition to receiving the word embedding, the input layer also adds a position vector for each input token. Then, the word and position embeddings are concatenated to form the final embeddings of the word, which enables the CNN-based models to perceive the word order like RNN and use the convolution module to convolution and nonlinear transformation of the embedding. In addition, to alleviate the problem of gradient disappearance and explosion, they introduced residual connections between layers. Their model achieves similar results to the RNN-based models on DUC-2004 and Gigaword datasets, and the training speed is greatly improved.

## CHAPTER 5

### CODING AND TESTING

```
import os
import re
import pickle
import string
import unicodedata
from random import randint

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from nltk.corpus import stopwords
from wordcloud import STOPWORDS, WordCloud

from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow.keras import Input, Model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.layers import LSTM, Bidirectional, Dense, Embedding,
from contractions import contractions_dict

for key, value in list(contractions_dict.items())[:10]:
    print(f'{key} == {value}')

# Using TPU

# detect and init the TPU
tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
tf.config.experimental_connect_to_cluster(tpu)
tf.tpu.experimental.initialize_tpu_system(tpu)

# instantiate a distribution strategy
tpu_strategy = tf.distribute.experimental.TPUStrategy(tpu)
filename1 = '../input/news-summary/news_summary.csv'
filename2 = '../input/news-summary/news_summary_more.csv'

df1 = pd.read_csv(filename1, encoding='iso-8859-1').reset_index(drop=True)
df2 = pd.read_csv(filename2, encoding='iso-8859-1').reset_index(drop=True)
df1.sample(5)
```

```

df2.sample(5)
df1_columns = df1.columns.tolist()
df1_columns.remove('headlines')
df1_columns.remove('text')
df1.drop(df1_columns, axis='columns', inplace=True)

df = pd.concat([df1, df2], axis='rows')
del df1, df2

# Shuffling the df
df = df.sample(frac=1).reset_index(drop=True)

print(f'Dataset size: {len(df)}')
df.sample(5)
Data preparation
def expand_contractions(text, contraction_map=contractions_dict):
    # Using regex for getting all contracted words
    contractions_keys = '|'.join(contraction_map.keys())
    contractions_pattern = re.compile(f'({contractions_keys})', flags=re.DOTALL)

    def expand_match(contraction):
        # Getting entire matched sub-string
        match = contraction.group(0)
        expanded_contraction = contraction_map.get(match)
        if not expanded_contraction:
            print(match)
            return match
        return expanded_contraction

    expanded_text = contractions_pattern.sub(expand_match, text)
    expanded_text = re.sub("''", "", expanded_text)
    return expanded_text

expand_contractions("y'all can't expand contractions i'd think")
# Converting to lowercase
df.text = df.text.apply(str.lower)
df.headlines = df.headlines.apply(str.lower)

df.sample(5)
df.headlines = df.headlines.apply(expand_contractions)
df.text = df.text.apply(expand_contractions)
df.sample(5)
# Remove punctuation from word
def rm_punc_from_word(word):
    clean_alphabet_list = [
        alphabet for alphabet in word if alphabet not in string.punctuation

```

```

]
return ''.join(clean_alphabet_list)

print(rm_punc_from_word('#cool!'))

# Remove punctuation from text
def rm_punc_from_text(text):
    clean_word_list = [rm_punc_from_word(word) for word in text]
    return ''.join(clean_word_list)

print(rm_punc_from_text("Frankly, my dear, I don't give a damn"))

# Remove numbers from text
def rm_number_from_text(text):
    text = re.sub('[0-9]+', '', text)
    return ' '.join(text.split()) # to rm `extra` white space

print(rm_number_from_text('You are 100times more sexier than me'))
print(rm_number_from_text('If you taught yes then you are 10 times more delusional than me'))

# Remove stopwords from text
def rm_stopwords_from_text(text):
    _stopwords = stopwords.words('english')
    text = text.split()
    word_list = [word for word in text if word not in _stopwords]
    return ' '.join(word_list)

rm_stopwords_from_text("Love means never having to say you're sorry")

# Cleaning text
def clean_text(text):
    text = text.lower()
    text = rm_punc_from_text(text)
    text = rm_number_from_text(text)
    text = rm_stopwords_from_text(text)

# there are hyphen(–) in many titles, so replacing it with empty str
# this hyphen(–) is different from normal hyphen(-)
text = re.sub('-', '', text)
text = ' '.join(text.split()) # removing `extra` white spaces

# Removing unnecessary characters from text
text = re.sub("(\t)", '', str(text)).lower()
text = re.sub("\r", '', str(text)).lower()
text = re.sub("\n", '', str(text)).lower()

# remove accented chars ('Sóme Āccēntēd tēxt' => 'Some Accented text')
text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode(

```



```
'utf-8', 'ignore')
)
```

```
text = re.sub("(__+)", ' ', str(text)).lower()
text = re.sub("(--+)", ' ', str(text)).lower()
text = re.sub("(~~+)", ' ', str(text)).lower()
text = re.sub("(\\+\\++)", ' ', str(text)).lower()
text = re.sub("(\\.\\.+) ", ' ', str(text)).lower()
```

```
text = re.sub(r"[<>()|&©ø\\[\\]\\\";,:?~*!]", ' ', str(text)).lower()
```

```
text = re.sub("(mailto:)", ' ', str(text)).lower()
text = re.sub(r"(\x9\d)", ' ', str(text)).lower()
text = re.sub("([iI][nN][cC]\d+)", 'INC_NUM', str(text)).lower()
text = re.sub("([cC][mM]\d+)|([cC][hH][gG]\d+)", 'CM_NUM',
              str(text)).lower()
```

```
text = re.sub("(\\.s+)", ' ', str(text)).lower()
text = re.sub("(\\-s+)", ' ', str(text)).lower()
text = re.sub("(\\:s+)", ' ', str(text)).lower()
text = re.sub("(\\s+\\.s+)", ' ', str(text)).lower()
```

```
try:
    url = re.search(r'((https?:\\*)([^\s]+))([^\s]+)', str(text))
    repl_url = url.group(3)
    text = re.sub(r'((https?:\\*)([^\s]+))([^\s]+)', repl_url, str(text))
except Exception as e:
    pass
```

```
text = re.sub("(\\s+)", ' ', str(text)).lower()
text = re.sub("(\\s+\\.s+)", ' ', str(text)).lower()
```

```
return text
```

```
clean_text("Mrs. Robinson, you're trying to seduce me, aren't you?")
```

```
'mrs robinson youre trying seduce arent'
```

```
df.text = df.text.apply(clean_text)
```

```
df.headlines = df.headlines.apply(clean_text)
```

```
df.sample(5)
```

```
# To customize colours of wordcloud texts
```

```
def wc_blue_color_func(word, font_size, position, orientation, random_state=None,
**kwargs):
```

```
    return "hsl(214, 67%%, %d%%)" % randint(60, 100)
```

```
# stopwords for wordcloud
```

```
def get_wc_stopwords():
```

```
# Adding words to stopwords
# these words showed up while plotting wordcloud for text
wc_stopwords.add('s')
wc_stopwords.add('one')
wc_stopwords.add('using')
wc_stopwords.add('example')
wc_stopwords.add('work')
wc_stopwords.add('use')
wc_stopwords.add('make')
```

```
# plot wordcloud
def plot_wordcloud(text, color_func):
    wc_stopwords = get_wc_stopwords()
    wc = WordCloud(stopwords=wc_stopwords, width=1200, height=600,
random_state=0).generate(text)

    f, axs = plt.subplots(figsize=(20, 10))
    with sns.axes_style("ticks"):
        sns.despine(offset=10, trim=True)
        plt.imshow(wc.recolor(color_func=color_func, random_state=0),
interpolation="bilinear")
        plt.xlabel('WordCloud')
```



```
text_count = [len(sentence.split()) for sentence in df.text]
```

```
headlines_count = [len(sentence.split()) for sentence in df.headlines]
```

```
pd.DataFrame({'text': text_count, 'headlines': headlines_count}).hist(bins=100,  
figsize=(16, 4), range=[0, 50])
```

```
plt.show()
```

```
# To check how many rows in a column has length (of the text) <= limit
```

```
def get_word_percent(column, limit):
```

```
    count = 0
```

```
    for sentence in column:
```

```
        if len(sentence.split()) <= limit:
```

```
            count += 1
```

```
    return round(count / len(column), 2)
```

```
# Check how many % of headlines have 0-13 words
```

```
print(get_word_percent(df.headlines, 13))
```

```
# Check how many % of summary have 0-42 words
```

```
print(get_word_percent(df.text, 42))
```

```
# select the summary and text between their defined max lens respectively
```

```
def trim_text_and_summary(df, max_text_len, max_summary_len):
```

```
    cleaned_text = np.array(df['text'])
```

```
    cleaned_summary = np.array(df['headlines'])
```

```
    short_text = []
```

```
    short_summary = []
```

```
    for i in range(len(cleaned_text)):
```

```
        if len(cleaned_text[i].split()) <= max_text_len and len(  
            cleaned_summary[i].split()
```

```
        ) <= max_summary_len:
```

```
            short_text.append(cleaned_text[i])
```

```
            short_summary.append(cleaned_summary[i])
```

```
    df = pd.DataFrame({'text': short_text, 'summary': short_summary})
```

```
    return df
```

```
df = trim_text_and_summary(df, max_text_len, max_summary_len)
```

```
print(f'Dataset size: {len(df)}')
```

```
df.sample(5)
```

```
# rare word analysis
```

```
def get_rare_word_percent(tokenizer, threshold):
```

```
    # threshold: if the word's occurrence is less than this then it's rare word
```

```
    count = 0
```

```

total_count = 0
frequency = 0
total_frequency = 0

for key, value in tokenizer.word_counts.items():
    total_count += 1
    total_frequency += value
    if value < threshold:
        count += 1
        frequency += value

return {
    'percent': round((count / total_count) * 100, 2),
    'total_coverage': round(frequency / total_frequency * 100, 2),
    'count': count,
    'total_count': total_count
}
# Splitting the training and validation sets
x_train, x_val, y_train, y_val = train_test_split(
    np.array(df['text']),
    np.array(df['summary']),
    test_size=0.1,
    random_state=1,
    shuffle=True
)
x_tokenizer = Tokenizer()
x_tokenizer.fit_on_texts(list(x_train))

x_tokens_data = get_rare_word_percent(x_tokenizer, 4)
print(x_tokens_data)
x_tokenizer = Tokenizer(num_words=x_tokens_data['total_count'] -
x_tokens_data['count'])
# else use this
x_tokenizer = Tokenizer()
x_tokenizer.fit_on_texts(list(x_train))
# save tokenizer
with open('x_tokenizer', 'wb') as f:
    pickle.dump(x_tokenizer, f, protocol=pickle.HIGHEST_PROTOCOL)
# one-hot-encoding
x_train_sequence = x_tokenizer.texts_to_sequences(x_train)
x_val_sequence = x_tokenizer.texts_to_sequences(x_val)

# padding upto max_text_len
x_train_padded = pad_sequences(x_train_sequence, maxlen=max_text_len,
padding='post')
x_val_padded = pad_sequences(x_val_sequence, maxlen=max_text_len, padding='post')

```

```

# if you're not using num_words parameter in Tokenizer then use this
x_vocab_size = len(x_tokenizer.word_index) + 1

# else use this
# x_vocab_size = x_tokenizer.num_words + 1

print(x_vocab_size)
y_tokenizer = Tokenizer()
y_tokenizer.fit_on_texts(list(y_train))

y_tokens_data = get_rare_word_percent(y_tokenizer, 6)
print(y_tokens_data)
y_tokenizer = Tokenizer(num_words=y_tokens_data['total_count'] -
y_tokens_data['count'])
# else use this
y_tokenizer = Tokenizer()
y_tokenizer.fit_on_texts(list(y_train))
# save tokenizer
with open('y_tokenizer', 'wb') as f:
    pickle.dump(y_tokenizer, f, protocol=pickle.HIGHEST_PROTOCOL)
# one-hot-encoding
y_train_sequence = y_tokenizer.texts_to_sequences(y_train)
y_val_sequence = y_tokenizer.texts_to_sequences(y_val)

# padding upto max_summary_len
y_train_padded = pad_sequences(y_train_sequence, maxlen=max_summary_len,
padding='post')
y_val_padded = pad_sequences(y_val_sequence, maxlen=max_summary_len,
padding='post')

# if you're not using num_words parameter in Tokenizer then use this
y_vocab_size = len(y_tokenizer.word_index) + 1

# else use this
# y_vocab_size = y_tokenizer.num_words + 1

print(y_vocab_size)
37524
# removing summary which only has sostok & eastok
def remove_indexes(summary_array):
    remove_indexes = []
    for i in range(len(summary_array)):
        count = 0
        for j in summary_array[i]:
            if j != 0:
                count += 1
        if count == 2:

```

```
        remove_indexes.append(i)
    return remove_indexes
```

```
remove_train_indexes = remove_indexes(y_train_padded)
remove_val_indexes = remove_indexes(y_val_padded)
```

```
y_train_padded = np.delete(y_train_padded, remove_train_indexes, axis=0)
x_train_padded = np.delete(x_train_padded, remove_train_indexes, axis=0)
```

```
y_val_padded = np.delete(y_val_padded, remove_val_indexes, axis=0)
x_val_padded = np.delete(x_val_padded, remove_val_indexes, axis=0)
```

```
latent_dim = 240
```

```
embedding_dim = 300
```

```
num_epochs = 50
```

```
def get_embedding_matrix(tokenizer, embedding_dim, vocab_size=None):
```

```
    word_index = tokenizer.word_index
```

```
    voc = list(word_index.keys())
```

```
    path_to_glove_file = '../input/glove6b/glove.6B.300d.txt'
```

```
    embeddings_index = {}
```

```
    with open(path_to_glove_file) as f:
```

```
        for line in f:
```

```
            word, coefs = line.split(maxsplit=1)
```

```
            coefs = np.fromstring(coefs, "f", sep=" ")
```

```
            embeddings_index[word] = coefs
```

```
    print("Found %s word vectors." % len(embeddings_index))
```

```
    num_tokens = len(voc) + 2 if not vocab_size else vocab_size
```

```
    hits = 0
```

```
    misses = 0
```

```
# Prepare embedding matrix
```

```
embedding_matrix = np.zeros((num_tokens, embedding_dim))
```

```
for word, i in word_index.items():
```

```
    embedding_vector = embeddings_index.get(word)
```

```
    if embedding_vector is not None:
```

```
        # Words not found in embedding index will be all-zeros.
```

```
        # This includes the representation for "padding" and "OOV"
```

```
        embedding_matrix[i] = embedding_vector
```

```
        hits += 1
```

```
    else:
```

```
        misses += 1
```

```
print("Converted %d words (%d misses)" % (hits, misses))
```

```
return embedding_matrix
```

```
x_embedding_matrix = get_embedding_matrix(x_tokenizer, embedding_dim,
x_vocab_size)
y_embedding_matrix = get_embedding_matrix(y_tokenizer, embedding_dim,
y_vocab_size)
Found 400000 word vectors.
Converted 55319 words (44456 misses)
Found 400000 word vectors.
Converted 27095 words (10428 misses)
print(x_embedding_matrix.shape)
print(y_embedding_matrix.shape)
def build_seq2seq_model_with_just_lstm(
    embedding_dim, latent_dim, max_text_len,
    x_vocab_size, y_vocab_size,
    x_embedding_matrix, y_embedding_matrix
):
    # instantiating the model in the strategy scope creates the model on the TPU
    with tpu_strategy.scope():

        # =====
        # 🔥 Encoder
        # =====
        encoder_input = Input(shape=(max_text_len, ))

        # encoder embedding layer
        encoder_embedding = Embedding(
            x_vocab_size,
            embedding_dim,
            embeddings_initializer=tf.keras.initializers.Constant(x_embedding_matrix),
            trainable=False
        )(encoder_input)

        # encoder lstm 1
        encoder_lstm1 = LSTM(
            latent_dim,
            return_sequences=True,
            return_state=True,
            dropout=0.4,
            recurrent_dropout=0.4
        )
        encoder_output1, state_h1, state_c1 = encoder_lstm1(encoder_embedding)

        # encoder lstm 2
        encoder_lstm2 = LSTM(
            latent_dim,
```

```

        return_sequences=True,
        return_state=True,
        dropout=0.4,
        recurrent_dropout=0.4
    )
    encoder_output, *encoder_final_states = encoder_lstm2(encoder_output1)

# =====
# 🌈 Decoder
# =====

# Set up the decoder, using `encoder_states` as initial state.

decoder_input = Input(shape=(None, ))

# decoder embedding layer
decoder_embedding_layer = Embedding(
    y_vocab_size,
    embedding_dim,
    embeddings_initializer=tf.keras.initializers.Constant(y_embedding_matrix),
    trainable=True
)
decoder_embedding = decoder_embedding_layer(decoder_input)

# decoder lstm 1
decoder_lstm = LSTM(
    latent_dim,
    return_sequences=True,
    return_state=True,
    dropout=0.4,
    recurrent_dropout=0.4
)
decoder_output, *decoder_final_states = decoder_lstm(
    decoder_embedding, initial_state=encoder_final_states
)

# dense layer
decoder_dense = TimeDistributed(
    Dense(y_vocab_size, activation='softmax')
)
decoder_output = decoder_dense(decoder_output)

# =====
# ⚡ Model
# =====
model = Model([encoder_input, decoder_input], decoder_output)
model.summary()

```



```

optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.001)
model.compile(
    optimizer=optimizer,
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

return {
    'model': model,
    'inputs': {
        'encoder': encoder_input,
        'decoder': decoder_input
    },
    'outputs': {
        'encoder': encoder_output,
        'decoder': decoder_output
    },
    'states': {
        'encoder': encoder_final_states,
        'decoder': decoder_final_states
    },
    'layers': {
        'decoder': {
            'embedding': decoder_embedding_layer,
            'last_decoder_lstm': decoder_lstm,
            'dense': decoder_dense
        }
    }
}

```

Seq2Seq model with Bidirectional LSTMs. Both encoder and decoder have Bidirectional LSTMs.

```

def build_seq2seq_model_with_bidirectional_lstm(
    embedding_dim, latent_dim, max_text_len,
    x_vocab_size, y_vocab_size,
    x_embedding_matrix, y_embedding_matrix
):
    # instantiating the model in the strategy scope creates the model on the TPU
    with tpu_strategy.scope():

        # =====
        # 🔥 Encoder
        # =====
        encoder_input = Input(shape=(max_text_len, ))

        # encoder embedding layer

```

```

encoder_embedding = Embedding(
    x_vocab_size,
    embedding_dim,
    embeddings_initializer=tf.keras.initializers.Constant(x_embedding_matrix),
    trainable=False,
    name='encoder_embedding'
)(encoder_input)


# encoder lstm1
encoder_bi_lstm1 = Bidirectional(
    LSTM(
        latent_dim,
        return_sequences=True,
        return_state=True,
        dropout=0.4,
        recurrent_dropout=0.4,
        name='encoder_lstm_1'
    ),
    name='encoder_bidirectional_lstm_1'
)
encoder_output1, forward_h1, forward_c1, backward_h1, backward_c1 =
encoder_bi_lstm1(
    encoder_embedding
)
encoder_bi_lstm1_output = [
    encoder_output1, forward_h1, forward_c1, backward_h1, backward_c1
]

# encoder lstm 2
encoder_bi_lstm2 = Bidirectional(
    LSTM(
        latent_dim,
        return_sequences=True,
        return_state=True,
        dropout=0.4,
        recurrent_dropout=0.4,
        name='encoder_lstm_2'
    ),
    name='encoder_bidirectional_lstm_2'
)
encoder_output2, forward_h2, forward_c2, backward_h2, backward_c2 =
encoder_bi_lstm2(
    encoder_output1
)
encoder_bi_lstm2_output = [
    encoder_output2, forward_h2, forward_c2, backward_h2, backward_c2
]

```

```

# encoder lstm 3
encoder_bi_lstm = Bidirectional(
    LSTM(
        latent_dim,
        return_sequences=True,
        return_state=True,
        dropout=0.4,
        recurrent_dropout=0.4,
        name='encoder_lstm_3'
    ),
    name='encoder_bidirectional_lstm_3'
)
encoder_output, *encoder_final_states = encoder_bi_lstm(encoder_output2)

# =====
#  Decoder
# =====

# Set up the decoder, using `encoder_states` as initial state.

decoder_input = Input(shape=(None, ))

# decoder embedding layer
decoder_embedding_layer = Embedding(
    y_vocab_size,
    embedding_dim,
    embeddings_initializer=tf.keras.initializers.Constant(y_embedding_matrix),
    trainable=False,
    name='decoder_embedding'
)
decoder_embedding = decoder_embedding_layer(decoder_input)

decoder_bi_lstm = Bidirectional(
    LSTM(
        latent_dim,
        return_sequences=True,
        return_state=True,
        dropout=0.4,
        recurrent_dropout=0.2,
        name='decoder_lstm_1'
    ),
    name='decoder_bidirectional_lstm_1'
)
decoder_output, *decoder_final_states = decoder_bi_lstm(
    decoder_embedding, initial_state=encoder_final_states
    # decoder_embedding, initial_state=encoder_final_states[:2]
)

```

```

) # taking only the forward states

# dense layer
decoder_dense = TimeDistributed(
    Dense(y_vocab_size, activation='softmax')
)
decoder_output = decoder_dense(decoder_output)

# =====
# ⚡ Model
# =====
model = Model([encoder_input, decoder_input], decoder_output,
name='seq2seq_model_with_bidirectional_lstm')
model.summary()

optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.001)
model.compile(
    optimizer=optimizer,
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

return {
    'model': model,
    'inputs': {
        'encoder': encoder_input,
        'decoder': decoder_input
    },
    'outputs': {
        'encoder': encoder_output,
        'decoder': decoder_output
    },
    'states': {
        'encoder': encoder_final_states,
        'decoder': decoder_final_states
    },
    'layers': {
        'decoder': {
            'embedding': decoder_embedding_layer,
            'last_decoder_lstm': decoder_bi_lstm,
            'dense': decoder_dense
        }
    }
}

```

Seq2Seq model with hybrid architecture. Here encoder has Bidirectional LSTMs while decoder has just LSTMs.

```

def build_hybrid_seq2seq_model(
    embedding_dim, latent_dim, max_text_len,
    x_vocab_size, y_vocab_size,
    x_embedding_matrix, y_embedding_matrix
):
    # instantiating the model in the strategy scope creates the model on the TPU
    with tpu_strategy.scope():

        # =====
        # 🔥 Encoder
        # =====
        encoder_input = Input(shape=(max_text_len, ))

        # encoder embedding layer
        encoder_embedding = Embedding(
            x_vocab_size,
            embedding_dim,
            embeddings_initializer=tf.keras.initializers.Constant(x_embedding_matrix),
            trainable=False,
            name='encoder_embedding'
        )(encoder_input)

        # encoder lstm1
        encoder_bi_lstm1 = Bidirectional(
            LSTM(
                latent_dim,
                return_sequences=True,
                return_state=True,
                dropout=0.4,
                recurrent_dropout=0.4,
                name='encoder_lstm_1'
            ),
            name='encoder_bidirectional_lstm_1'
        )
        encoder_output1, forward_h1, forward_c1, backward_h1, backward_c1 =
encoder_bi_lstm1(
    encoder_embedding
)
        encoder_bi_lstm1_output = [
            encoder_output1, forward_h1, forward_c1, backward_h1, backward_c1
        ]

        # encoder lstm 2
        encoder_bi_lstm2 = Bidirectional(
            LSTM(
                latent_dim,
                return_sequences=True,

```

```

        return_state=True,
        dropout=0.4,
        recurrent_dropout=0.4,
        name='encoder_lstm_2'
    ),
    name='encoder_bidirectional_lstm_2'
)
encoder_output2, forward_h2, forward_c2, backward_h2, backward_c2 =
encoder_bi_lstm2(
    encoder_output1
)
encoder_bi_lstm2_output = [
    encoder_output2, forward_h2, forward_c2, backward_h2, backward_c2
]

# encoder lstm 3
encoder_bi_lstm = Bidirectional(
    LSTM(
        latent_dim,
        return_sequences=True,
        return_state=True,
        dropout=0.4,
        recurrent_dropout=0.4,
        name='encoder_lstm_3'
    ),
    name='encoder_bidirectional_lstm_3'
)
encoder_output, *encoder_final_states = encoder_bi_lstm(encoder_output2)

# =====
# 🌈 Decoder
# =====

# Set up the decoder, using `encoder_states` as initial state.

decoder_input = Input(shape=(None, ))

# decoder embedding layer
decoder_embedding_layer = Embedding(
    y_vocab_size,
    embedding_dim,
    embeddings_initializer=tf.keras.initializers.Constant(y_embedding_matrix),
    trainable=False,
    name='decoder_embedding'
)
decoder_embedding = decoder_embedding_layer(decoder_input)

```

```

decoder_lstm = LSTM(
    latent_dim,
    return_sequences=True,
    return_state=True,
    dropout=0.4,
    recurrent_dropout=0.2,
    name='decoder_lstm_1'
)
decoder_output, *decoder_final_states = decoder_lstm(
    decoder_embedding, initial_state=encoder_final_states[:2]
) # taking only the forward states

# dense layer
decoder_dense = TimeDistributed(
    Dense(y_vocab_size, activation='softmax')
)
decoder_output = decoder_dense(decoder_output)

# =====
# ⚡ Model
# =====
model = Model([encoder_input, decoder_input], decoder_output,
name='seq2seq_model_with_bidirectional_lstm')
model.summary()

optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.001)
model.compile(
    optimizer=optimizer,
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

return {
    'model': model,
    'inputs': {
        'encoder': encoder_input,
        'decoder': decoder_input
    },
    'outputs': {
        'encoder': encoder_output,
        'decoder': decoder_output
    },
    'states': {
        'encoder': encoder_final_states,
        'decoder': decoder_final_states
    },
    'layers': {

```

```

        'decoder': {
            'embedding': decoder_embedding_layer,
            'last_decoder_lstm': decoder_lstm,
            'dense': decoder_dense
        }
    }
}

seq2seq = build_seq2seq_model_with_just_lstm(
    embedding_dim, latent_dim, max_text_len,
    x_vocab_size, y_vocab_size,
    x_embedding_matrix, y_embedding_matrix
)
model = seq2seq['model']

encoder_input = seq2seq['inputs']['encoder']
decoder_input = seq2seq['inputs']['decoder']

encoder_output = seq2seq['outputs']['encoder']
decoder_output = seq2seq['outputs']['decoder']

encoder_final_states = seq2seq['states']['encoder']
decoder_final_states = seq2seq['states']['decoder']

decoder_embedding_layer = seq2seq['layers']['decoder']['embedding']
last_decoder_lstm = seq2seq['layers']['decoder']['last_decoder_lstm']
decoder_dense = seq2seq['layers']['decoder']['dense']
history = model.fit(
    [x_train_padded, y_train_padded[:, :-1]],
    y_train_padded.reshape(y_train_padded.shape[0], y_train_padded.shape[1], 1)[:, 1:],
    epochs=num_epochs,
    batch_size=128 * tpu_strategy.num_replicas_in_sync,
    callbacks=callbacks,
    validation_data=(
        [x_val_padded, y_val_padded[:, :-1]],
        y_val_padded.reshape(y_val_padded.shape[0], y_val_padded.shape[1], 1)[:, 1:]
    )
)
# Loss
plt.plot(history.history['loss'][1:], label='train loss')
plt.plot(history.history['val_loss'], label='val')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='lower right')
# Next, let's build the dictionary to convert the index to word for target and source
vocabulary:
reverse_target_word_index = y_tokenizer.index_word
reverse_source_word_index = x_tokenizer.index_word

```



```

target_word_index = y_tokenizer.word_index
def build_seq2seq_model_with_just_lstm_inference(
    max_text_len, latent_dim, encoder_input, encoder_output,
    encoder_final_states, decoder_input, decoder_output,
    decoder_embedding_layer, decoder_dense, last_decoder_lstm
):
    # Encode the input sequence to get the feature vector
    encoder_model = Model(
        inputs=encoder_input, outputs=[encoder_output] + encoder_final_states
    )

    # Decoder setup
    # Below tensors will hold the states of the previous time step
    decoder_state_input_h = Input(shape=(latent_dim, ))
    decoder_state_input_c = Input(shape=(latent_dim, ))
    decoder_hidden_state_input = Input(shape=(max_text_len, latent_dim))

    # Get the embeddings of the decoder sequence
    decoder_embedding = decoder_embedding_layer(decoder_input)

    # To predict the next word in the sequence, set the initial
    # states to the states from the previous time step
    decoder_output, *decoder_states = last_decoder_lstm(
        decoder_embedding,
        initial_state=[decoder_state_input_h, decoder_state_input_c]
    )

    # A dense softmax layer to generate prob dist. over the target vocabulary
    decoder_output = decoder_dense(decoder_output)

    # Final decoder model
    decoder_model = Model(
        [decoder_input] + [decoder_hidden_state_input, decoder_state_input_h,
decoder_state_input_c],
        [decoder_output] + decoder_states
    )

    return (encoder_model, decoder_model)
def build_seq2seq_model_with_bidirectional_lstm_inference(
    max_text_len, latent_dim, encoder_input, encoder_output,
    encoder_final_states, decoder_input, decoder_output,
    decoder_embedding_layer, decoder_dense, last_decoder_bi_lstm
):
    # Encode the input sequence to get the feature vector
    encoder_model = Model(
        inputs=encoder_input, outputs=[encoder_output] + encoder_final_states

```

```

)

# Decoder setup
# Below tensors will hold the states of the previous time step
decoder_state_forward_input_h = Input(shape=(latent_dim, ))
decoder_state_forward_input_c = Input(shape=(latent_dim, ))
decoder_state_backward_input_h = Input(shape=(latent_dim, ))
decoder_state_backward_input_c = Input(shape=(latent_dim, ))

# Create the hidden input layer with twice the latent dimension,
# since we are using bi - directional LSTM's we will get
# two hidden states and two cell states
decoder_hidden_state_input = Input(shape=(max_text_len, latent_dim * 2))

decoder_initial_state = [
    decoder_state_forward_input_h, decoder_state_forward_input_c,
    decoder_state_backward_input_h, decoder_state_backward_input_c
]

# Get the embeddings of the decoder sequence
decoder_embedding = decoder_embedding_layer(decoder_input)

# To predict the next word in the sequence, set the initial
# states to the states from the previous time step
decoder_output, *decoder_states = last_decoder_bi_lstm(
    decoder_embedding, initial_state=decoder_initial_state
)

# A dense softmax layer to generate prob dist. over the target vocabulary
decoder_output = decoder_dense(decoder_output)

# Final decoder model
decoder_model = Model(
    [decoder_input] + [decoder_hidden_state_input] + decoder_initial_state,
    [decoder_output] + decoder_states
)

return (encoder_model, decoder_model)
def build_hybrid_seq2seq_model_inference(
    max_text_len, latent_dim, encoder_input, encoder_output,
    encoder_final_states, decoder_input, decoder_output,
    decoder_embedding_layer, decoder_dense, last_decoder_bi_lstm
):

# Encode the input sequence to get the feature vector
encoder_model = Model(
    inputs=encoder_input, outputs=[encoder_output] + encoder_final_states

```

```

)

# Decoder setup
# Below tensors will hold the states of the previous time step
decoder_state_forward_input_h = Input(shape=(latent_dim, ))
decoder_state_forward_input_c = Input(shape=(latent_dim, ))
# decoder_state_backward_input_h = Input(shape=(latent_dim, ))
# decoder_state_backward_input_c = Input(shape=(latent_dim, ))

# Create the hidden input layer with twice the latent dimension,
# since we are using bi - directional LSTM's we will get
# two hidden states and two cell states
decoder_hidden_state_input = Input(shape=(max_text_len, latent_dim * 2))

decoder_initial_state = [
    decoder_state_forward_input_h, decoder_state_forward_input_c,
    #decoder_state_backward_input_h, decoder_state_backward_input_c
]

# Get the embeddings of the decoder sequence
decoder_embedding = decoder_embedding_layer(decoder_input)

# To predict the next word in the sequence, set the initial
# states to the states from the previous time step
decoder_output, *decoder_states = last_decoder_bi_lstm(
    decoder_embedding, initial_state=decoder_initial_state
)

# A dense softmax layer to generate prob dist. over the target vocabulary
decoder_output = decoder_dense(decoder_output)

# Final decoder model
decoder_model = Model(
    [decoder_input] + [decoder_hidden_state_input] + decoder_initial_state,
    [decoder_output] + decoder_states
)

return (encoder_model, decoder_model)
encoder_model, decoder_model = build_seq2seq_model_with_just_lstm_inference(
    max_text_len, latent_dim, encoder_input, encoder_output,
    encoder_final_states, decoder_input, decoder_output,
    decoder_embedding_layer, decoder_dense, last_decoder_lstm
)
encoder_model.summary()
def decode_sequence_seq2seq_model_with_just_lstm(
    input_sequence, encoder_model, decoder_model
):

```

```

# Encode the input as state vectors.
e_out, e_h, e_c = encoder_model.predict(input_sequence)

# Generate empty target sequence of length 1.
target_seq = np.zeros((1, 1))

# Populate the first word of target sequence with the start word.
target_seq[0, 0] = target_word_index[start_token]

stop_condition = False
decoded_sentence = ""

while not stop_condition:
    output_tokens, h, c = decoder_model.predict(
        [target_seq] + [e_out, e_h, e_c]
    )

    # Sample a token
    sampled_token_index = np.argmax(output_tokens[0, -1, :])
    sampled_token = reverse_target_word_index[sampled_token_index]

    if sampled_token != end_token:
        decoded_sentence += ' ' + sampled_token

    # Exit condition: either hit max length or find stop word.
    if (sampled_token == end_token) or (len(decoded_sentence.split()) >=
(max_summary_len - 1)):
        stop_condition = True

    # Update the target sequence (of length 1).
    target_seq = np.zeros((1, 1))
    target_seq[0, 0] = sampled_token_index

    # Update internal states
    e_h, e_c = h, c

return decoded_sentence

def decode_sequence_seq2seq_model_with_bidirectional_lstm(
    input_sequence, encoder_model, decoder_model
):
    # Encode the input as state vectors.
    e_out, *state_values = encoder_model.predict(input_sequence)

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1, 1))

    # Populate the first word of target sequence with the start word.

```

```

target_seq[0, 0] = target_word_index[start_token]

stop_condition = False
decoded_sentence = ""

while not stop_condition:
    output_tokens, *decoder_states = decoder_model.predict(
        [target_seq] + [e_out] + state_values
    )

    # Sample a token
    sampled_token_index = np.argmax(output_tokens[0, -1, :]) # Greedy Search
    sampled_token = reverse_target_word_index[sampled_token_index + 1]

    if sampled_token != end_token:
        decoded_sentence += ' ' + sampled_token

    # Exit condition: either hit max length or find stop word.
    if (sampled_token == end_token) or (len(decoded_sentence.split()) >=
(max_summary_len - 1)):
        stop_condition = True

    # Update the target sequence (of length 1).
    target_seq = np.zeros((1, 1))
    target_seq[0, 0] = sampled_token_index

    # Update internal states
    state_values = decoder_states

return decoded_sentence

def decode_sequence_hybrid_seq2seq_model(
    input_sequence, encoder_model, decoder_model
):
    # Encode the input as state vectors.
    e_out, *state_values = encoder_model.predict(input_sequence)

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1, 1))

    # Populate the first word of target sequence with the start word.
    target_seq[0, 0] = target_word_index[start_token]

    stop_condition = False
    decoded_sentence = ""

    while not stop_condition:
        output_tokens, *decoder_states = decoder_model.predict(

```

```

    [target_seq] + [e_out] + state_values[:2]
)

# Sample a token
sampled_token_index = np.argmax(output_tokens[0, -1, :]) # Greedy Search
sampled_token = reverse_target_word_index[sampled_token_index + 1]

if sampled_token != end_token:
    decoded_sentence += ' ' + sampled_token

# Exit condition: either hit max length or find stop word.
if (sampled_token == end_token) or (len(decoded_sentence.split()) >=
(max_summary_len - 1)):
    stop_condition = True

# Update the target sequence (of length 1).
target_seq = np.zeros((1, 1))
target_seq[0, 0] = sampled_token_index

# Update internal states
state_values = decoder_states

return decoded_sentence
def seq2summary(input_sequence):
    new_string = ""
    for i in input_sequence:
        if (
            (i != 0 and i != target_word_index[start_token]) and
            (i != target_word_index[end_token])
        ):
            new_string = new_string + reverse_target_word_index[i] + ' '
    return new_string
def seq2text(input_sequence):
    new_string = ""
    for i in input_sequence:
        if i != 0:
            new_string = new_string + reverse_source_word_index[i] + ' '
    return new_string
l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

if len(l) % 3 != 0:
    while len(l) % 3 != 0:
        l.append(0)
print(l)

lst_i = 3
for i in range(0, len(l), 3):

```

```

print(l[i:i + lst_i])

print(' '.join(['', 'james', 'ethan', '', 'tony']))
print(' '.join(' '.join(['', 'james', 'ethan', '', 'tony']).split()))
def predict_text(text, decode_sequence, encoder_model, decoder_model):
    original_text = text
    text = clean_text([text]) # generator
    text_list = original_text.split()

    if len(text_list) <= max_text_len:
        text = expand_contractions(text)
        text = clean_text(text)
        text = f'_START_ {text} _END_'
        text = f'{start_token} {text} {end_token}'

        seq = x_tokenizer.texts_to_sequences([' '.join(text_list)])
        padded = pad_sequences(seq, maxlen=max_text_len, padding='post')
        pred_summary = decode_sequence(
            padded.reshape(1, max_text_len), encoder_model, decoder_model
        )
        return pred_summary
    else:
        pred_summary = ""

    # breaking long texts to individual max_text_len texts and predicting on them
    while len(text_list) % max_text_len == 0:
        text_list.append("")

    lst_i = max_text_len
    for i in range(0, len(text_list), max_text_len):
        _text_list = original_text.split()[i:i + lst_i]
        _text = ' '.join(_text_list)
        _text = ' '.join(
            _text.split()
        ) # to remove spaces that were added to make len(text_list) % max_text_len == 0

        _text = expand_contractions(_text)
        _text = clean_text(_text) # generator
        _text = f'_START_ {_text} _END_'
        _text = f'{start_token} {_text} {end_token}'
        # print(_text, '\n')

        _seq = x_tokenizer.texts_to_sequences([_text])
        _padded = pad_sequences(_seq, maxlen=max_text_len, padding='post')
        _pred = decode_sequence(
            _padded.reshape(1, max_text_len), encoder_model, decoder_model
        )

```

```

        pred_summary += ' ' + ''.join(_pred.split()[1:-2])
        pred_summary = ''.join(pred_summary.split())

    return pred_summary
# Testing on training data
for i in range(0, 15):
    print(f"# {i+1} News: ", seq2text(x_train_padded[i]))
    print("Original summary: ", seq2summary(y_train_padded[i]))
    print(
        "Predicted summary: ",
        decode_sequence_seq2seq_model_with_just_lstm(
            x_train_padded[i].reshape(1, max_text_len), encoder_model,
            decoder_model
        )
    )
    print()
# Testing on validation data
for i in range(0, 15):
    print(f"# {i+1} News: ", seq2text(x_val_padded[i]))
    print("Original summary: ", seq2summary(y_val_padded[i]))
    print(
        "Predicted summary: ",
        decode_sequence_seq2seq_model_with_just_lstm(
            x_val_padded[i].reshape(1, max_text_len), encoder_model,
            decoder_model
        )
    )
    print()
models_info = {
    'just_lstm': {
        'model': build_seq2seq_model_with_just_lstm,
        'inference': build_seq2seq_model_with_just_lstm_inference,
        'decode_sequence': decode_sequence_seq2seq_model_with_just_lstm
    },
    'bidirectional_lstm': {
        'model': build_seq2seq_model_with_bidirectional_lstm,
        'inference': build_seq2seq_model_with_bidirectional_lstm_inference,
        'decode_sequence': decode_sequence_seq2seq_model_with_bidirectional_lstm
    },
    'hybrid_model': {
        'model': build_hybrid_seq2seq_model,
        'inference': build_hybrid_seq2seq_model_inference,
        'decode_sequence': decode_sequence_hybrid_seq2seq_model
    }
}
Model with just LSTMs

model_func = models_info['just_lstm']['model']

```



```

inference_func = models_info['just_lstm']['inference']
decode_sequence_func = models_info['just_lstm']['decode_sequence']
seq2seq = model_func(
    embedding_dim, latent_dim, max_text_len,
    x_vocab_size, y_vocab_size,
    x_embedding_matrix, y_embedding_matrix
)

model = seq2seq['model']

encoder_input = seq2seq['inputs']['encoder']
decoder_input = seq2seq['inputs']['decoder']

encoder_output = seq2seq['outputs']['encoder']
decoder_output = seq2seq['outputs']['decoder']

encoder_final_states = seq2seq['states']['encoder']
decoder_final_states = seq2seq['states']['decoder']

decoder_embedding_layer = seq2seq['layers']['decoder']['embedding']
last_decoder_lstm = seq2seq['layers']['decoder']['last_decoder_lstm']
decoder_dense = seq2seq['layers']['decoder']['dense']

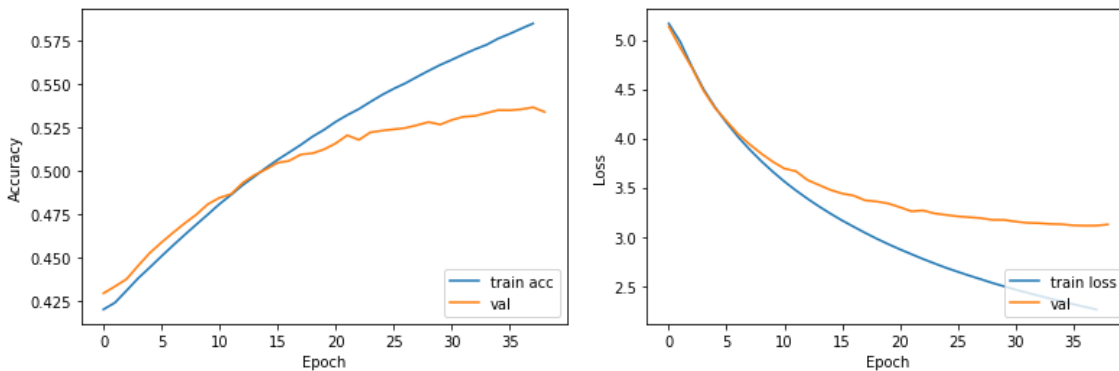
model.summary()
history = model.fit(
    [x_train_padded, y_train_padded[:, :-1]],
    y_train_padded.reshape(y_train_padded.shape[0], y_train_padded.shape[1], 1)[:, 1:],
    epochs=num_epochs,
    batch_size=128 * tpu_strategy.num_replicas_in_sync,
    callbacks=callbacks,
    validation_data=(
        [x_val_padded, y_val_padded[:, :-1]],
        y_val_padded.reshape(y_val_padded.shape[0], y_val_padded.shape[1], 1)[:, 1:]
    )
)

```

## CHAPTER 6

### RESULTS

From the results, we can know that the large-scale language model based on pretraining has achieved the current SOTA results. This is expected, because these pretrained models are pretrained on a large-scale external corpus (e.g., Wikipedia) to capture deeper semantic information of natural language. And now, the models based on pretraining almost dominate the list of various NLP tasks. However, the pretraining process requires enormous computing resources and massive data to support it. Most researchers can only use pretrained models to fine-tune for adapting to specific tasks. Compared with the Seq2Seq baseline, adding pointers and coverage mechanisms can significantly improve the quality of the generated summary. Furthermore, adding internal guidance information can better control the generation process of the ABS systems, such as keywords, key sentences, etc., which allows the model to focus more on the important parts of the document when decoding, thereby enhancing the informativeness of generated summaries. In addition, the introduction of external information into the system can also further enrich the semantic information of the model, thereby ensuring the readability and factual correctness of the generated summary, such as commonsense knowledge graphs. In particular, the introduction of triples improves the factual correctness and the ROUGE score of generated summaries. Compared with the baseline models, the use of reinforcement learning training strategy further enhances the performance of summarization systems.



## CHAPTER 7

### CONCLUSION AND FUTURE ENHANCEMENTS

#### **i. Conclusion**

In this paper, we provide a comprehensive overview of currently available abstractive text summarization models. We show the overall framework of the ABS systems based on neural networks, the details of model design, training strategies, and summarize the advantages and disadvantages of these methods. We also introduced some datasets and evaluation metrics that are widely used in the field of text summarization. Finally, we report the performance analysis results of different models on large-scale datasets, which should be helpful for researchers to choose a suitable framework and model according to their own needs. We hope that our work can provide some new perspectives and inspirations for the future research and application of ABS.

#### **ii. Future work**

With the amount of data becoming more extensive and the attributes of the data becoming more and more abundant, the ABS models based on deep learning have great potential. However, the existing ABS methods have many limitations, which are the future challenges and research directions of the research community. These challenges will help the researchers to identify areas where further research is needed. We discuss several directions worth studying in the future, as follows:

- (1) **Personalized Summary Generation.** At present, most of the summary models are based on input documents and do not consider the subjective demands of users. A system that can generate personalized summaries according to specific user needs will be very useful in e-commerce and text-based recommendation.
- (2) **Introduce Richer External Knowledge.** Both models guided by keywords (sentences) and models enhanced by factual triples essentially use knowledge from within the document. However, with the development of knowledge graph technology, a lot of commonsense knowledge can be used to enhance the model and further improve the factual correctness of the generated summaries.
- (3) **Flexible Stopping Criteria during the Generation Process.** The generation of a summary is an iterative process. At present, almost all methods limit the maximum length of summary in advance to terminate this process. However, in fact, different scenarios and fields, and even different input documents, have different lengths of the summary. For

example, the summary of a scientific article is longer than a news article. How to make the system adaptively terminate the iterative process is a significant research direction.

(4) Comprehensive Evaluation Metrics. Evaluating the quality of generated summary either automatically or manually is a difficult task. Manual evaluation is highly subjective and can only be performed on a small testing set, which is not statistically significant. However, the current automatic evaluation is difficult to consider the semantic level. Therefore, a new comprehensive automatic evaluation metric is essential, which can not only help evaluate the quality of a summary but also support the training process of the ABS system.

(5) Cross-Language or Low-Resource Language Summarization. Currently, popular public summarization datasets are based on English. Using these publicly available large-scale English datasets to train a cross-language summarization model to generate summaries in low-resource languages is an interesting and meaningful work. This research is still in its infancy and requires more researchers to work together to make a breakthrough [150].

## REFERENCES

1. A. Khan, M. A. Gul, M. Zareei et al., “Movie Review Summarization Using Supervised Learning and Graph-Basedranking Algorithm,” *Computational intelligence and neuroscience*, vol. 2020, Article ID 7526580, 2020.  
View at: [Publisher Site](#) | [Google Scholar](#)
2. G. C. V. Vilca and M. A. S. Cabezudo, “A study of abstractive summarization using semantic representations and discourse level information,” *Text, Speech, and Dialogue*, pp. 482–490, 2017.  
View at: [Publisher Site](#) | [Google Scholar](#)
3. W. S. El-Kassas, C. R. Salama, A. A. Rafea, and H. K. Mohamed, “Automatic text summarization: a comprehensive survey,” *Expert Systems with Applications*, vol. 165, Article ID 113679, 2021.  
View at: [Publisher Site](#) | [Google Scholar](#)
4. K. Shen, P. Hao, and R. Li, “A Compressive Sensing Model for Speeding up Text Classification,” *Computational Intelligence and Neuroscience*, Article ID 8879795, 2020.  
View at: [Publisher Site](#) | [Google Scholar](#)
5. S. Tao, C. Shen, L. Zhu, and D. Tao, “SVD-CNN: A Convolutional Neural Network Model with Orthogonal Constraints Based on SVD for Context-Aware Citation Recommendation,” *Computational Intelligence and Neuroscience*, Article ID 5343214, 2020.  
View at: [Publisher Site](#) | [Google Scholar](#)
6. Y. Zhu, W. Zheng, and H. Tang, “Interactive Dual Attention Network for Text Sentiment Classification,” *Computational Intelligence and Neuroscience*, Article ID 8858717, 2020.  
View at: [Publisher Site](#) | [Google Scholar](#)
7. J. Dan and H. Jin, “Text Semantic Classification of Long Discourses Based on Neural Networks with Improved Focal Loss,” *Computational Intelligence and Neuroscience*, Article ID 8845362, 2021.  
View at: [Publisher Site](#) | [Google Scholar](#)
8. R. Rzepka, S. Takishita, and K. Araki, “Language Model-based context augmentation for world knowledge bases,” in *Proceedings of the 34th Annual Conference of the Japanese Society for Artificial Intelligence*, June 2020.  
View at: [Google Scholar](#)
9. P. Dybala, M. Yatsu, M. Ptaszynski, R. Rafal, and A. Kenji, “Towards joking, humor sense equipped and emotion aware conversational Systems,” *Advances in Affective and Pleasurable Design*, vol. 483, pp. 657–669, 2016.  
View at: [Google Scholar](#)

10. M. Mohamed and M. Oussalah, "SRL-ESA-TextSum: a text summarization approach based on semantic role labeling and explicit semantic analysis," *Information Processing & Management*, vol. 56, no. 4, pp. 1356–1372, 2019.  
View at: [Publisher Site](#) | [Google Scholar](#)
11. C. Barros, E. Lloret, E. Saquete, and B. Navarro-Colorado, "NATSUM: narrative abstractive summarization through cross-document timeline generation," *Information Processing & Management*, vol. 56, no. 5, pp. 1775–1793, 2019.  
View at: [Publisher Site](#) | [Google Scholar](#)
12. S. Hou, Y. Huang, C. Fei, S. Zhang, and R. Lu, "Holographic lexical chain and its application in Chinese text summarization," *Web and Big Data*, pp. 266–281, 2017.  
View at: [Publisher Site](#) | [Google Scholar](#)
13. E. Chu and P. Liu, "MeanSum: a neural model for unsupervised multi-document abstractive summarization," in *Proceedings of the 36th International Conference on Machine Learning*, pp. 1223–1232, California, USA, June 2019.  
View at: [Google Scholar](#)
14. M. Gambhir and V. Gupta, "Recent automatic text summarization techniques: a survey," *Artificial Intelligence Review*, vol. 47, no. 1, pp. 1–66, 2017.  
View at: [Publisher Site](#) | [Google Scholar](#)
15. J. Cheng and M. Lapata, "Neural Summarization by Extracting Sentences and Words. Preprint," Article ID 07252, <https://arxiv.org/abs/1603.07252>.  
View at: [Google Scholar](#)
16. Y. Gao, Y. Wang, L. Liu, Y. Guo, and H. Huang, "Neural abstractive summarization fusing by global generative topics," *Neural Computing & Applications*, vol. 32, no. 9, pp. 5049–5058, 2019.  
View at: [Publisher Site](#) | [Google Scholar](#)
17. S. Li and J. Xu, "A two-step abstractive summarization model with asynchronous and enriched-information decoding," *Neural Computing & Applications*, vol. 33, no. 4, pp. 1159–1170, 2021.  
View at: [Publisher Site](#) | [Google Scholar](#)
18. Z. Liang, J. Du, and C. Li, "Abstractive social media text summarization using selective reinforced Seq2Seq attention model," *Neurocomputing*, vol. 410, pp. 432–440, 2020.  
View at: [Publisher Site](#) | [Google Scholar](#)
19. L. Lebanoff, K. Song, and F. Liu, "Adapting the neural encoder-decoder framework from single to multi-document summarization," in *Proceeding of the Conference on Empirical Methods in Natural Language Processing*, pp. 4131–4141, Brussels, Belgium, 2018.  
View at: [Publisher Site](#) | [Google Scholar](#)

20. J. Zhu, Q. Wang, and Y. Wang, “NCLS: neural cross-lingual summarization,” in *Proceeding of the Conference on Empirical Methods in Natural Language Processing*, pp. 3045–3055, Hong Kong, China, 2019.  
View at: [Publisher Site](#) | [Google Scholar](#)
21. R. Nallapati, F. Zhai, and B. Zhou, “SummaRuNNer: a recurrent neural network based sequence model for extractive summarization of documents,” in *Proceeding of the 31th AAAI Conference on Artificial Intelligence*, pp. 3075–3081, San Francisco, USA.  
View at: [Google Scholar](#)
22. Y. Liu and M. Lapata, “Hierarchical transformers for multi-document summarization,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 5070–5081, Florence, Italy, 2019.  
View at: [Publisher Site](#) | [Google Scholar](#)
23. T. Cai, M. Shen, H. Peng, L. Jiang, and Q. Dai, “Improving transformer with sequential context representations for abstractive text summarization,” *Natural Language Processing and Chinese Computing*, pp. 512–524, 2019.  
View at: [Publisher Site](#) | [Google Scholar](#)
24. E. Linhares, S. Huet, J. M. Torres, and A. C. Linhares, “Compressive approaches for cross-language multi-document summarization,” *Data & Knowledge Engineering*, vol. 125, Article ID 101763, 2020.  
View at: [Publisher Site](#) | [Google Scholar](#)
25. F. Xu, Z. Pan, and R. Xia, “E-commerce product review sentiment classification based on a naïve Bayes continuous learning framework,” *Information Processing & Management*, vol. 57, no. 5, Article ID 102221, 2020.  
View at: [Publisher Site](#) | [Google Scholar](#)
26. Z. Chan, Y. Zhang, X. Chen, and S. Gao, “Selection and generation: learning towards multi-product advertisement post generation,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 3818–3829.  
View at: [Publisher Site](#) | [Google Scholar](#)
27. H. Xu, W. Wang, X. Mao, X. Jiang, and M. Lan, “Scaling up open tagging from tens to thousands: comprehension empowered attribute value extraction from product title,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 5214–5223, Florence, Italy, July 2019.  
View at: [Google Scholar](#)
28. Q. Chen, J. Lin, Y. Zhang, H. Yang, Z. Jingren, and T. Jie, “Towards knowledge-based personalized product description generation in e-commerce,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3040–3050,

Washington, DC, USA, July 2019.

View at: [Publisher Site](#) | [Google Scholar](#)

29. V. Daultani, L. Nio, and Y. J. Chung, “Unsupervised extractive summarization for product description using coverage maximization with attribute concept,” in *Proceedings of the IEEE 13th International Conference on Semantic Computing*, pp. 114–117, Newport Beach, CA, USA, March 2019.

View at: [Publisher Site](#) | [Google Scholar](#)

30. Y. Gong, X. Luo, K. Q. Zhu, and W. Ou, “Automatic generation of Chinese short product titles for mobile display,” in *Proceedings of the 33th AAAI Conference on Artificial Intelligence*, pp. 9460–9465, Hawaii, USA, July 2019.

View at: [Publisher Site](#) | [Google Scholar](#)

31. J. Wang, J. Tian, L. Qiu, and L. Sheng, “A multi-task learning approach for improving product title compression with user search log data,” in *Proceedings of the 32th AAAI Conference on Artificial Intelligence*, pp. 451–458, New Orleans, USA, January 2018.

View at: [Google Scholar](#)

32. C. Zhu, Z. Yang, R. Gmyr, M. Zeng, and X. Huan, “Make lead Bias in Your Favor: A Simple and Effective Method for News Summarization,” <https://arxiv.org/abs/1605.09065>.

View at: [Google Scholar](#)

33. X. Zhang, F. Wei, and M. Zhou, “HIBERT: document level pre-training of hierarchical bidirectional transformers for document summarization,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 5059–5069, Florence, Italy, 2019.

View at: [Google Scholar](#)

34. J. Wang, Y. Hou, J. Liu, Y. Cao, and C. Y. Lin, “A statistical framework for product description generation,” in *Proceedings of the Eighth International Joint Conference on Natural Language Processing*, pp. 187–192, Taiwan, China.

View at: [Google Scholar](#)

35. H. P. Luhn, “The automatic creation of literature abstracts,” *IBM Journal of Research and Development*, vol. 2, no. 2, pp. 159–165, 1958.

View at: [Publisher Site](#) | [Google Scholar](#)

36. M. T. Maybury, “Generating summaries from event data,” *Information Processing & Management*, vol. 31, no. 5, pp. 735–751, 1995.

View at: [Publisher Site](#) | [Google Scholar](#)

37. D. R. Radev, E. Hovy, and K. McKeown, “Introduction to the special issue on summarization,” *Computational Linguistics*, vol. 28, no. 4, pp. 399–408, 2002.

View at: [Publisher Site](#) | [Google Scholar](#)

38. D. Chen, Z. Ma, L. Wei, M. Jinlin, and Z. Yanbin, “MTQA: Text-Based Multitype Question and



Answer Reading Comprehension Model,” *Computational Intelligence and Neuroscience*, Article ID 8810366, 2021.

View at: [Publisher Site](#) | [Google Scholar](#)

39. A. Khan, A. Gul, M. Zareei et al., “Movie Review Summarization Using Supervised Learning and Graph-Based Ranking Algorithm,” *Computational intelligence and neuroscience*, vol. 2020, Article ID 7526580, 2020.

View at: [Publisher Site](#) | [Google Scholar](#)

40. Y. Dong, Y. Shen, and E. Crawford, “BanditSum: extractive summarization as a contextual bandit,” in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3739–3748, Brussels, Belgium, 2018.

View at: [Publisher Site](#) | [Google Scholar](#)

41. X. Zhang, M. Lapata, F. Wei et al., “Neural latent extractive document summarization,” in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 779–784, Brussels, Belgium.

View at: [Publisher Site](#) | [Google Scholar](#)

42. Z. Deng, F. Ma, R. Huang, W. Luo, and X. Luo, “A Two-stage Chinese text summarization algorithm using keyword information and adversarial learning,” *Neurocomputing*, vol. 425, pp. 117–126, 2021.

View at: [Publisher Site](#) | [Google Scholar](#)

43. J. Zheng, Z. Zhao, M. Yang, M. Xiao, J. Yan, and X. Yan, “Abstractive meeting summarization by hierarchical adaptive segmental network learning with multiple revising steps,” *Neurocomputing*, vol. 378, pp. 179–188, 2020.

View at: [Publisher Site](#) | [Google Scholar](#)

44. S. Takase, J. Suzuki, and N. Okazaki, “Neural headline generation on abstract meaning representation,” in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1054–1059, Texas, USA.

View at: [Publisher Site](#) | [Google Scholar](#)

45. P. Mehta and P. Majumder, “Effective aggregation of various summarization techniques effective aggregation of various summarization techniques,” *Information Processing & Management*, vol. 54, no. 2, pp. 145–158, 2018.

View at: [Publisher Site](#) | [Google Scholar](#)

46. J. Tan, X. Wan, and J. Xiao, “From neural sentence summarization to headline generation: a coarse-to-fine approach,” in *Proceeding of the 2017 International Joint Conference on Artificial Intelligence*, pp. 4109–4115.

View at: [Google Scholar](#)

47. A. Dlikman and M. Last, “Using machine learning methods and linguistic features in

single-document extractive summarization,” in *Proceeding of DMNLP@ PKDD/ECML*, pp. 1–8, Riva del Garda, Italy.

View at: [Google Scholar](#)

48. R. Nallapati, B. Zhou, and M. Ma, “Classify or Select: Neural Architectures for Extractive Document Summarization,” 2016, <https://arxiv.org/abs/1611.04244>.

View at: [Google Scholar](#)

49. A. Cohan, F. Dernoncourt, D. S. Kim, B. Trung, and K. Seokhwan, “A discourse-aware attention model for abstractive summarization of long documents,” in *Proceeding of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 615–621, New Orleans, USA.

View at: [Google Scholar](#)

50. C. Li, W. Xu, S. Li, and G. Sheng, “Guiding generation for abstractive text summarization based on key information guide network,” in *Proceeding of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 55–60, New Orleans, USA.

View at: [Google Scholar](#)

51. B. Sankaran, H. Mi, Y. Al-Onaizan, and I. Abe, “Temporal attention model for neural machine translation,” 2016, <https://arxiv.org/abs/1608.02927>.

View at: [Google Scholar](#)

52. J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159.

View at: [Google Scholar](#)

53. C. Napoles, M. Gormley, and B. V. Durme, “Annotated Gigaword. Proceeding of the Joint Workshop on Automatic Knowledge Base Construction and Web-Scale Knowledge Extraction,” pp. 95–100, 2012.

View at: [Google Scholar](#)

54. G. Erkan and D. R. Radev, “LexRank: graph-based lexical centrality as salience in text summarization,” *Journal of Artificial Intelligence Research*, vol. 22, pp. 457–479, 2004.

View at: [Publisher Site](#) | [Google Scholar](#)

55. C. Y. Lin and E. Hovy, “The automated acquisition of topic signatures for text summarization,” in *Proceedings of the 18th Conference on Computational Linguistics. Stroudsburg*, pp. 495–501, Association for Computational Linguistics, KunMing, China, July 2000.

View at: [Publisher Site](#) | [Google Scholar](#)

56. D. R. Radev, H. Jing, M. Styś, and D. Tam, “Centroid-based summarization of multiple documents,” *Information Processing & Management*, vol. 40, no. 6, pp. 919–938, 2004.

View at: [Publisher Site](#) | [Google Scholar](#)

57. H. Li, J. Zhu, J. Zhang, C. Zong, and X. He, “Keywords-guided abstractive sentence

summarization,” in *Proceedings of the 2020 AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, pp. 8196–8203, Seattle, WA, USA.

View at: [Publisher Site](#) | [Google Scholar](#)

58. Y. Zhang, D. Merck, and E. Tsai, “Optimizing the factual correctness of a summary: a study of summarizing radiology reports,” in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

View at: [Publisher Site](#) | [Google Scholar](#)

59. L. Miao, D. Cao, J. Li, and W. Guan, “Multi-modal product title compression,” *Information Processing & Management*, vol. 57, no. 1, Article ID 102123, 2020.

View at: [Publisher Site](#) | [Google Scholar](#)

60. F. M. Belém, R. M. Silva, C. M. V. de Andrade et al., “Fixing the curse of the bad product descriptions—search-boosted tag recommendation for E-commerce products,” *Information Processing & Management*, vol. 57, no. 5, Article ID 102289, 2020.

View at: [Publisher Site](#) | [Google Scholar](#)

61. L. K. Ramasamy, S. Kadry, Y. Nam, and M. N. Meqdad, “Performance analysis of sentiments in Twitter dataset using SVM models,” *International Journal of Electrical and Computer Engineering*, vol. 11, no. 3, p. 2275, 2021.

View at: [Publisher Site](#) | [Google Scholar](#)

62. “Mining product innovation ideas from online reviews,” *Information Processing & Management*, vol. 58, no. 1, Article ID 102389.

View at: [Google Scholar](#)

63. T. de Melo, A. S. da Silva, E. S. de Moura, and P. Calado, “OpinionLink: leveraging user opinions for product catalog enrichment,” *Information Processing & Management*, vol. 56, no. 3, pp. 823–843, 2019.

View at: [Publisher Site](#) | [Google Scholar](#)

64. Z. Luo, S. Huang, and K. Q. Zhu, “Knowledge empowered prominent aspect extraction from product reviews,” *Information Processing & Management*, vol. 56, no. 3, pp. 408–423, 2019.

View at: [Publisher Site](#) | [Google Scholar](#)

65. M. R. Mane, S. Kedia, A. Mantha, S. Guo, and K. Achan, “Product Title Generation for Conversational Systems Using BERT,” <https://arxiv.org/>.

View at: [Google Scholar](#)

66. J. G. C. de Souza, M. Kozielski, P. Mathur, and E. Chang, “Generating e-commerce product titles and predicting their quality,” in *Proceedings of the 11th International Conference on Natural Language Generation*, pp. 233–243, Brussels, Belgium.

View at: [Google Scholar](#)

67. A. M. Rush, S. Chopra, and J. Weston, “A neural attention model for abstractive sentence

summarization,” in *Proceedings 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 379–389, Boston, MA, USA.

View at: [Publisher Site](#) | [Google Scholar](#)

68. J. P. Cheng, L. Dong, and M. Lapata, “Long short-term memory networks for machine reading,” in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 551–561, Las Vegas, NV, USA.

View at: [Publisher Site](#) | [Google Scholar](#)

69. K. Cho, B. Merriënboer, C. Gulcehre et al., “Learning phrase representations using RNN encoder-decoder for statistical machine translation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pp. 1724–1734, Doha, Qatar, 2014.

View at: [Publisher Site](#) | [Google Scholar](#)

70. Y. Kim, “Convolutional neural networks for sentence classification,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pp. 1746–1751, Doha, Qatar, 2014.

View at: [Publisher Site](#) | [Google Scholar](#)

71. K. Xu, W. Hu, J. Leskovec, and J. Stefanie, “How powerful are graph neural networks,” in *Proceedings of the 7th International Conference on Learning Representations*, New Orleans, USA, 2019.

View at: [Google Scholar](#)

72. M. A. Ranzato, S. Chopra, M. Auli, and W. Zaremba, “Sequence level training with recurrent neural networks,” in *Proceedings of the 4th International Conference on Learning Representations*, Puerto Rico, 2016.

View at: [Google Scholar](#)

73. S. Chopra, M. Auli, and A. M. Rush, “Abstractive sentence summarization with attentive recurrent neural networks,” in *Proceedings 2016 Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 93–98, Human Language Technologies, California.

View at: [Publisher Site](#) | [Google Scholar](#)

74. K. Lopyrev, “Generating News Headlines with Recurrent Neural Networks,” 2015, <https://arxiv.org/abs/1512.01712>.

View at: [Google Scholar](#)

75. Q. Chen, X. D. Zhu, Z. H. Ling, W. Si, and J. Hui, “Distraction-based neural networks for modeling document,” in *Proceedings of the 25 International Joint Conference on Artificial Intelligence*, pp. 2754–2760, 2016.

View at: [Google Scholar](#)

76. R. Nallapati, B. Zhou, C. Gulcehre, and X. Bing, “Abstractive text summarization using

sequence-to-sequence rnns and beyond,” in *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pp. 280–290, Berlin, Germany, 2016.

View at: [Google Scholar](#)

77. R. Paulus, C. Xiong, and R. Socher, “A deep reinforced model for abstractive summarization,” in *Proceedings of 6th International Conference on Learning Representations*, Vancouver, Canada, 2018.

View at: [Google Scholar](#)

78. R. Lin, S. Liu, and M. Yang, “Hierarchical recurrent neural network for document modelling,” in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 899–907, Lisbon, Portugal.

View at: [Google Scholar](#)

79. Z. Yang, D. Yang, and C. Dyer, “Hierarchical attention networks for document classification,” in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1480–1489, San Diego, CA, USA.

View at: [Google Scholar](#)

80. J. Li, M. T. Luong, and D. Jurafsky, “A hierarchical neural autoencoder for paragraphs and documents,” in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing*, pp. 1106–1115, Beijing, China, 2015.

View at: [Publisher Site](#) | [Google Scholar](#)

81. A. Jadhav and V. Rajan, “Extractive summarization with swap-net: sentences and words from alternating pointer networks,” in *Proceedings of the 56th annual meeting of the association for computational linguistics*, pp. 142–151, Melbourne, Australia, 2018.

View at: [Publisher Site](#) | [Google Scholar](#)

82. J. Tan, X. Wan, and J. Xiao, “Abstractive document summarization with a graph-based attentional neural model,” in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pp. 1171–1181, Vancouver, Canada, 2017.

View at: [Publisher Site](#) | [Google Scholar](#)

83. W. Li, X. Xiao, and Y. Lyu, “Improving neural abstractive document summarization with structural regularization,” in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 4078–4087, Brussels, Belgium.

View at: [Publisher Site](#) | [Google Scholar](#)

84. W. T. Hsu, C. K. Lin, and M. Y. Lee, “A unified model for extractive and abstractive summarization using inconsistency loss,” in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pp. 132–141, Melbourne, Australia, 2018.

View at: [Publisher Site](#) | [Google Scholar](#)

85. L. Wang, J. L. Yao, Y. Z. Tao, L. Zhong, W. Liu, and Q. Du, “A reinforced topic-aware convolutional sequence-to-sequence model for abstractive text summarization,” in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pp. 4453–4460, Stockholm, Sweden, 2018.

View at: [Publisher Site](#) | [Google Scholar](#)

86. Y. Zhang, D. Shen, and G. Wang, “Deconvolutional paragraph representation learning,” in *Proceedings of the 31th International Conference on Neural Information Processing Systems*, pp. 4172–4182, California, USA, 2017.

View at: [Google Scholar](#)

87. J. Gehring, M. Auli, and D. Grangier, “A convolutional encoder model for neural machine translation,” in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pp. 123–135, Vancouver, Canada, 2017.

View at: [Publisher Site](#) | [Google Scholar](#)

88. J. Gehring, M. Auli, and D. Grangier, “Convolutional sequence to sequence learning,” in *Proceedings of International Conference on Machine Learning*, pp. 1243–1252, Sydney, Australia, 2017.

View at: [Google Scholar](#)

89. A. Fan, D. Grangier, and M. Auli, “Controllable abstractive summarization,” in *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pp. 45–54, Melbourne, Australia, 2018.

View at: [Google Scholar](#)

90. S. Narayan, S. B. Cohen, and M. Lapata, “Don’t give me the details, just the summary! Topic-aware convolutional neural networks for extreme summarization,” in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1797–1807, Brussels, Belgium, 2018.

View at: [Google Scholar](#)

91. D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent dirichlet allocation,” *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

View at: [Google Scholar](#)

92. C. Gulcehre, S. Ahn, and R. Nallapati, “Pointing the unknown words,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pp. 140–149, Berlin, Germany, 2016.

View at: [Google Scholar](#)

93. S. Jean, K. Cho, and R. Memisevic, “On using very large target vocabulary for neural machine translation,” in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pp.



1–10, Beijing, China, 2015.

View at: [Publisher Site](#) | [Google Scholar](#)

94. J. Gu, Z. Lu, and H. Li, “Incorporating copying mechanism in sequence-to-sequence learning,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pp. 1631–1640, Berlin, German, 2016.

View at: [Publisher Site](#) | [Google Scholar](#)

95. A. See, P. J. Liu, and C. D. Manning, “Get to the point: summarization with pointer-generator networks,” in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pp. 1073–1083, Vancouver, Canada, 2017.

View at: [Publisher Site](#) | [Google Scholar](#)

96. X. Shen, Y. Zhao, and H. Su, “Improving latent alignment in text summarization by generalizing the pointer generator,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 3762–3773, Hong Kong, China.

View at: [Publisher Site](#) | [Google Scholar](#)

97. B. Goodrich, V. Rao, and P. J. Liu, “Assessing the factual accuracy of generated text,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 166–175, New York, USA, 2019.

View at: [Publisher Site](#) | [Google Scholar](#)

98. T. Falke, L. F. R. Ribeiro, and P. A. Utama, “Ranking generated summaries by correctness: an interesting but challenging application for natural language inference,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2214–2220, Florence, Italy, 2019.

View at: [Publisher Site](#) | [Google Scholar](#)

99. W. Kryscinski, B. McCann, and C. Xiong, “Evaluating the factual consistency of abstractive text summarization,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 9332–9346, Punta Cana, Dominican Republic.

View at: [Publisher Site](#) | [Google Scholar](#)

100. Z. Cao, F. Wei, and W. Li, “Faithful to the original: fact aware neural abstractive summarization,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, Louisiana, USA, 2018.

View at: [Publisher Site](#) | [Google Scholar](#)

101. G. Angeli, M. J. J. Premkumar, and C. D. Manning, “Leveraging linguistic structure for open domain information extraction,” in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pp. 344–354, Beijing, China, 2015.

View at: [Publisher Site](#) | [Google Scholar](#)

102. H. Li, J. Zhu, and J. Zhang, “Ensure the correctness of the summary: incorporate entailment knowledge into abstractive sentence summarization,” in *Proceedings of the 27th International Conference on Computational Linguistics*, pp. 1430–1441, New Mexico, USA, 2018.

View at: [Google Scholar](#)

103. J. Bos and K. Markert, “Recognising textual entailment with logical inference,” in *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pp. 628–635, New Mexico, USA, 2005.

View at: [Publisher Site](#) | [Google Scholar](#)

104. M. Norouzi, S. Bengio, and N. Jaitly, “Reward augmented maximum likelihood for neural structured prediction,” *Advances in Neural Information Processing Systems*, vol. 29, pp. 1723–1731, 2016.

View at: [Google Scholar](#)

105. C. Zhu, W. Hinthorn, and R. Xu, “Boosting Factual Correctness of Abstractive Summarization with Knowledge Graph,” Article ID 08612, 2020, <https://arxiv.org/>.

View at: [Google Scholar](#)

106. P. Veličković, G. Cucurull, A. Casanova, R. Adriana, L. Pietro, and B. Yoshua, *Graph Attention Networks*, International Conference on Learning Representations, Vancouver Canada, 2018.

107. M. Zhang, G. Zhou, W. Yu, and W. Liu, “FAR-ASS: fact-aware reinforced abstractive sentence summarization,” *Information Processing & Management*, vol. 58, no. 3, Article ID 102478, 2021.

View at: [Publisher Site](#) | [Google Scholar](#)

108. H. Lin and V. Ng, “Abstractive summarization: a survey of the state of the art,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 9815–9822, 2019.

View at: [Publisher Site](#) | [Google Scholar](#)

109. J. Xu and G. Durrett, “Neural extractive text summarization with syntactic compression,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 3292–3303.

View at: [Publisher Site](#) | [Google Scholar](#)

110. K. M. Hermann, T. Kocisky, E. Grefenstette et al., “Teaching machines to read and comprehend,” in *Proceedings of the Annual Conference on Neural Information Processing Systems*, pp. 1693–1701, 2015.

View at: [Google Scholar](#)

111. G. Durrett, T. Berg-Kirkpatrick, and D. Klein, “Learning-based single-document



summarization with compression and anaphoricity constraints,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pp. 1998–2008, 2016.

View at: [Publisher Site](#) | [Google Scholar](#)

112. B. Hu, Q. Chen, and F. Zhu, “LCSTS: a large scale Chinese short text summarization dataset,” in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1967–1972, 2015.

View at: [Publisher Site](#) | [Google Scholar](#)

113. A. R. Fabbri, I. Li, and T. She, “Multi-news: a large-scale multi-document summarization dataset and abstractive hierarchical model,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 1074–1084, 2019.

View at: [Publisher Site](#) | [Google Scholar](#)

114. J. Carletta, S. Ashby, S. Bourban, and F. Mike, *The AMI Meeting Corpus: A Pre-announcement*, International workshop on machine learning for multimodal interaction, Edinburgh, UK, pp. 28–39, 2005.

View at: [Publisher Site](#)

115. Y. Fang, H. Zhu, and E. Muszyńska, “A proposition-based abstractive summariser,” in *Proceedings of COLING 2016, The 26th International Conference on Computational Linguistics*, pp. 567–578, Technical Papers.

View at: [Google Scholar](#)

116. M. Yasunaga, J. Kasai, R. Zhang et al., “Scisummnet: a large annotated corpus and content-impact models for scientific paper summarization with citation networks,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 7386–7393, 2019.

View at: [Publisher Site](#) | [Google Scholar](#)

117. C. Lin, “ROUGE: a package for automatic evaluation of summaries,” *Proceedings of the Workshop on Text Summarization Branches Out*, pp. 74–81, 2004.

View at: [Google Scholar](#)

118. N. Schluter, “The limits of automatic summarisation according to ROUGE,” in *Proceedings of the 15th Conference of the European*, pp. 41–45, Chapter of the Association for Computational Linguistics, Valencia, Spain, 2017.

View at: [Publisher Site](#) | [Google Scholar](#)

119. Q. Zhou, N. Yang, and F. Wei, “Selective encoding for abstractive sentence summarization,” in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pp. 1095–1104, Vancouver, Canada, 2017.

View at: [Publisher Site](#) | [Google Scholar](#)

120. P. Li, W. Lam, and L. Bing, “Deep recurrent generative decoder for abstractive text

summarization,” in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 2091–2100, Copenhagen, Denmark.

View at: [Publisher Site](#) | [Google Scholar](#)

121. A. Vaswani, N. Shazeer, N. Parmar et al., “Attention is all you need,” *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

View at: [Google Scholar](#)

122. K. Song, L. Zhao, and F. Liu, “Structure-infused copy mechanisms for abstractive summarization,” in *Proceedings of the 27th International Conference on Computational Linguistics*, pp. 1717–1729, New-Mexico, USA, 2018.

View at: [Google Scholar](#)

123. H. Guo, R. Pasunuru, and M. Bansal, “Soft layer-specific multi-task summarization with entailment and question generation,” in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pp. 687–697, Melbourne, Australia, 2018.

View at: [Publisher Site](#) | [Google Scholar](#)

124. J. Lin, X. Sun, and S. Ma, “The progress of asthma management in China,” in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, vol. 41, no. 3, pp. 163–165, Melbourne, Australia, 2018.

View at: [Publisher Site](#) | [Google Scholar](#)

125. R. K. Amplayo, S. Lim, and S. Hwang, “Entity commonsense representation for neural abstractive summarization,” in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 697–707, Human Language Technologies, New Orleans, Louisiana, 2018.

View at: [Publisher Site](#) | [Google Scholar](#)

126. Z. Cao, W. Li, and S. Li, “Retrieve, rerank and rewrite: soft template based neural summarization,” in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pp. 152–161, New Orleans, Louisiana, 2018.

View at: [Publisher Site](#) | [Google Scholar](#)

127. K. Song, L. Lebanoff, Q. Guo et al., “Joint parsing and generation for abstractive summarization,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, pp. 8894–8901, New York, USA, 2020.

View at: [Publisher Site](#) | [Google Scholar](#)

128. W. Wang, Y. Gao, and H. Y. Huang, “Concept pointer network for abstractive summarization,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 3076–3085, Hong Kong, China, 2019.

View at: [Publisher Site](#) | [Google Scholar](#)

129. K. Song, X. Tan, T. Qin, J. Lu, and T. Liu, “Mass: Masked Sequence to Sequence Pre-training for Language Generation,” 2019, <https://arxiv.org/abs/1905.02450>.  
View at: [Google Scholar](#)
130. L. Dong, N. Yang, W. Wang, and F. Wei, “Unified Language Model Pre-training for Natural Language Understanding and Generation,” 2019, <https://arxiv.org/abs/1905.03197>.  
View at: [Google Scholar](#)
131. K. Wang, X. Quan, and R. Wang, “BiSET: Bi-directional selective encoding with template for abstractive summarization,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2153–2162, Florence, Italy, 2019.  
View at: [Publisher Site](#) | [Google Scholar](#)
132. J. Zhang, Y. Zhao, M. Saleh, and P. Liu, “Pegasus: pre-training with extracted gap-sentences for abstractive summarization,” in *Proceedings of the International Conference on Machine Learning*, pp. 11328–11339, Vienna, Austria, 2020.  
View at: [Google Scholar](#)
133. D. Xiao, H. Zhang, and Y. Li, “Ernie-gen: an enhanced multi-flow pre-training and fine-tuning framework for natural language generation,” in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pp. 3997–4003, Yokohama, Japan, 2020.  
View at: [Publisher Site](#) | [Google Scholar](#)
134. W. Qi, Y. Yan, Y. Gong et al., “ProphetNet: predicting future N-gram for sequence-to-sequence pre-training,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pp. 2401–2410, 2020.  
View at: [Publisher Site](#) | [Google Scholar](#)
135. A. Aghajanyan, A. Shrivastava, A. Gupta, and N. Goyal, “Better fine-tuning by reducing representational collapse,” 2020, Punta Cana, Dominican Republic, <https://arxiv.org/abs/2008.03156>.  
View at: [Google Scholar](#)
136. Z. Fan, Y. Gong, D. Liu, Z. Wei, S. Wang, and J. Jiao, “Mask attention networks: rethinking and strengthen transformer,” in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1692–1701, Mexico City, Mexico, 2021.  
View at: [Publisher Site](#) | [Google Scholar](#)
137. S. Takase and S. Kiyono, “Rethinking perturbations in encoder-decoders for fast training,” in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5767–5780, Mexico City, Mexico, 2021.

View at: [Publisher Site](#) | [Google Scholar](#)

138. A. Aghajanyan, A. Gupta, A. Shrivastava, and X. Chen, “Muppet: massive multi-task representations with pre-finetuning,” 2021, <https://arxiv.org/pdf/2101.11038.pdf>.

View at: [Google Scholar](#)

139. R. Pasunuru and M. Bansal, “Multi-reward reinforced summarization with saliency and entailment,” in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 646–653, Human Language Technologies, New Orleans, Louisiana.

View at: [Publisher Site](#) | [Google Scholar](#)

140. Y. C. Chen and M. Bansal, “Fast abstractive summarization with reinforce-selected sentence rewriting,” in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pp. 675–686, Melbourne, Australia, 2018.

View at: [Publisher Site](#) | [Google Scholar](#)

141. S. Gehrmann, Y. Deng, and A. M. Rush, “Bottom-up abstractive summarization,” in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 4098–4109, Brussels, Belgium, 2018.

View at: [Publisher Site](#) | [Google Scholar](#)

142. A. Celikyilmaz, A. Bosselut, and X. He, “Deep communicating agents for abstractive summarization,” in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 1662–1675, Human Language Technologies, New Orleans, Louisiana, 2018.

View at: [Google Scholar](#)

143. E. Moroshko, G. Feigenblat, and H. Roitman, “An editorial network for enhanced document summarization,” in *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pp. 57–63, Hong Kong, China, 2019.

View at: [Publisher Site](#) | [Google Scholar](#)

144. H. Zhang, J. Cai, and J. Xu, “Pretraining-based natural language generation for text summarization,” in *Proceedings of the 23rd Conference on Computational Natural Language Learning*, pp. 789–797, Hong Kong, China, 2019.

View at: [Publisher Site](#) | [Google Scholar](#)

145. Y. Liu and M. Lapata, “Text summarization with pretrained encoders,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 3730–3740, Hong Kong, China, 2019.

View at: [Publisher Site](#) | [Google Scholar](#)

146. M. Lewis, Y. Liu, and N. Goyal, “BART: denoising sequence-to-sequence pre-training for

natural language generation, translation, and comprehension,” in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 7871–7880, Seattle, WA, USA, 2020.

View at: [Publisher Site](#) | [Google Scholar](#)

147. H. Bao, L. Dong, F. Wei, and W. Wang, “Unilmv2: pseudo-masked language models for unified language model pre-training,” in *Proceedings of the International Conference on Machine Learning*, pp. 642–652, Vienna , Austria, 2020.

View at: [Google Scholar](#)

148. X. Liang, L. Wu, J. Li, Y. Wang, Q. Meng, and T. Qin, “R-drop: Regularized Dropout for Neural Networks,” 2021, <https://arxiv.org/pdf/2106.14448.pdf>.

View at: [Google Scholar](#)

149. Z. Du, Y. Qian, X. Liu, M. Ding, J. Qiu, and Z. Yang, “All NLP tasks are generation tasks: a general pretraining framework,” 2021, <https://arxiv.org/abs/2103.10360>.

View at: [Google Scholar](#)

150. M. Reid, E. Marrese-Taylor, and Y. Matsuo, “Subformer: A Parameter Reduced Transformer,” *Machel Reid, Edison Marrese-Taylor, Yutaka Matsuo*, 2020.

View at: [Google Scholar](#)