Automated Text Summarization
A Review and Recommendations

Steven Shearing, Abigail Gertner, Benjamin Wellner, Liz Merkhofer

November 2020

MITRE Technical Report
Project Name: Machine Intelligence for Language Understanding (MILU)
Project Number: 01MSRD21-CA
Department Number: Human Language Technologies (L133)
Location: McLean, VA

Contents

List of Figures

List of Tables

Executive Summary

This report presents an examination of a wide variety of automatic summarization models. We broadly assign summarization models into two overarching categories: extractive and abstractive summarization. Extractive summarization essentially reduces the summarization problem to a subset selection problem by returning portions of the input as the summary. This allows extractive models to be simpler, as they do not require the ability to generate new language. Within the extractive models, we focus primarily on graph-based solutions and recurrent neural networks; however, this is not an exhaustive search of all extractive solutions. Optimization based systems such as submodular optimization [20] [21] [9] and semantic volume maximization [48], or other pre-graph systems [25] which can be viable solutions were left outside the scope of this report.

For abstractive systems, we examine recurrent neural networks and their successor the Transformer. Unlike extractive systems, abstractive methods generate entirely new text given the input to be summarized, much like a human might do. As a result, these models often generate summaries which appear more natural. This comes at the cost of increased model capacity requirements, as the model must learn how to generate new text, not only identify important text. This can also lead to larger failures, as a failure can now not only occur in text identification, but also text generation. As with extractive systems, the models we examine do not represent the entirety of all abstractive solutions; however, our choice of models for both extractive and abstractive summarization cover a majority of the current solution space.

Overall, we find a scaling trade-off between computational resources versus summarization quality as we progress from the early graph models to the most recent Transformer models. Graph algorithms are unsupervised, requiring little to no training data and are often usable right out of the box with no specialized hardware requirements. However, they have a low performance ceiling, and will generally not be able to match the quality of summaries generated by later models. Despite this low ceiling, the unsupervised nature of the algorithm makes it a desirable solution for a number of problem domains, particularly in cases where the user does not have in-domain labeled training data.

On the other hand, Transformer models have extremely high output quality, but have a number of requirements that make them difficult to use in general. Transformer models must be trained on large amounts of unlabeled data to learn strong language generation, and they must be fine-tuned on labeled summarization data, which may not exist for many domains. Due to their immense size, specialized GPU hardware is required to train and run these models. If the particular use case manages to meet all the requirements of a Transformer, it is highly recommended to use one.

Recurrent neural networks provide a middle ground to the graph algorithms and Transformer model. While not fully unsupervised, recurrent neural networks require much less training data than Transformers, often being able to train on datasets that are simply too small to be usable for a Transformer. While recurrent neural networks also require specialized GPU hardware to train, they are less computationally expensive and can be trained faster, with cheaper and smaller GPUs, and may not require a GPU to run inference in a fast enough manner.

1 Introduction

Language understanding is central to many of the largest problems in information processing. MITRE's sponsors are faced with overwhelming quantities of free text data and lack the tools necessary to turn this data into actionable information. Increasingly, one of the most essential problems analysts and decision makers face is not just how to find the information they are already looking for, but what information they should be looking for in the first place. Due to the large quantity of data, it is not feasible for these individuals to read and search through the entire body of text to identify the events, trends, or themes that may demand further action.

One potential tool for addressing the large data issue is text summarization. Instead of forcing analysts to personally search through long text themselves for information, we can apply machine learning algorithms to the documents within a corpus and reduce them to only a few representative sentences. This reduction provides two advantages: analysts only need to consume the reduced text, which can be orders of magnitude smaller, and the text they do consume is more likely to be immediately relevant. While the summarized document will contain fewer facts and details, which for some use-cases may make it less valuable on its own,

it can serve as a signal to further examine the source document or skip it entirely. This provides a valuable time saving service.

In fact, summarization as a large data reduction mechanism is already performed by humans across a variety of domains. Scientific publications include an abstract, a few sentences which serve as an overview of what the publication contains. News articles often contain highlights, either at the start or end of the article, which reiterate the article's most important facts. Even the titles of documents can be seen as a form of summary, designed to inform the reader of the contents and convince them to read further. These natural summaries provide ample data to train new state-of-the-art neural network models to automatically perform novel summarization: generating new language to succinctly encapsulate the original document. This form of summarization is abstractive summarization.

Another form of summarization, which has been historically more common in practice, is extractive summarization. Rather than mimic human behavior and generate new sentences, extractive systems pull out sentences directly from the source document to serve as the summary. This allows the system to be much simpler, as it only needs to learn how to identify important or central information, unlike abstractive systems which must also learn how to model language. Due to the simpler nature of the problem, until very recently extractive systems were the standard solution to machine generated summaries.

2 Evaluating Summarization Systems

In order to properly evaluate the various machine summarization algorithms and systems, one must understand the available evaluation metrics and their shortcomings. At their core, automatic evaluation metrics measure the quality of the summarization system output. This is inherently a subjective task – there is no singular correct answer; present a document to five different humans, and all five will produce a different summary. In fact, beyond a general notion of capturing all the key elements of the original text in a coherent output, there is arguably no way to objectively quantify a 'good' summary. Present those five human summaries to a new group of human evaluators, and the evaluators will all disagree on the exact quality score each summary should receive.

Thus, automatic evaluation metrics are not measured against an impossible objective quality score, but rather against their correlation with human judgement. An evaluation metric is calibrated if it gives high scores to output that humans have scored high, and low scores to output that humans have scored low. If the evaluation metric scores one summary higher than another, the expectation is that humans will also in general score that first summary higher as well. Furthermore, it is not imperative that the automatic metric and humans agree on exact scores, though it is beneficial that they are close; rather, it is only vital that they agree on relative scores. While human judgement is still subjective by nature, it is the chosen measure as presumably humans are the end-users for the generated summaries.

There are two primary automatic metrics reported across summarization models: ROUGE [19] and METEOR [2]. Both metrics compare generated summaries against reference summaries. ROUGE, which is actually a set of related metrics, most commonly measures the overlap of unigrams, bigrams, and longest common subsequence between the generated summary and the reference (R1, R2, and RL respectively), though other lesser-used metrics are included in the ROUGE group. METEOR measures the harmonic mean of precision and recall of individual words (unigrams) between the generated summary and reference, with a higher weight on recall. Both directly compare whether or not the individual generated tokens appears in the reference.

This direct comparison is problematic for a number of reasons. First, it requires exact token match—but multiple words can have similar meanings or be interchangeable. It also does not consider word order, sentence structure, or other language features that can make similar output look different, or different output look similar. Aside from the pitfalls of how the metric is computed across the generation and reference, there's the fundamental problem that it relies on a specific reference at all. Comparing against one specific summary only measures how close the generation was to one possible output, which is not guaranteed to be high quality itself. For this reason, these metrics cannot be used to measure the quality of individual summaries, but only in the aggregate to compare systems based on their performance on a large number of test summaries.

Despite these issues, both ROUGE and METEOR are the standard reported metrics as they have been

the most correlated to human judgement while remaining cheap to compute. However, this is only true historically—when the average quality of summarization systems has been lower, the direct comparison to specific references was more highly correlated with human judgement. This is likely because the average quality was low enough that getting closer to one potential summary was enough to increase human judgement. As summarization system quality has increased though, correlation between these metrics and human judgement has decreased [37]. Problematically, metrics start to disagree on which candidate generations are better, and it is unclear which metric is most trustworthy.

Despite the gradual decorrelation witnessed as summarization systems have improved, ROUGE and METEOR are still the standard metrics reported today. While new automatic metrics, such as BERTScore [51], have been created which seek to help improve human judgement correlations, they have not been adopted into widespread use. Human evaluation methods, like Pyramid [33], have also been designed, but are very expensive and slow to compute due to their reliance on humans. Until one of these new metrics proves itself as a new standard, evaluating cutting edge systems will be inconsistent. For the remainder of this report, we exclusively use ROUGE score (R1, R2, etc.) to avoid some of these inconsistencies.

3 Extractive Summarization

3.1 Graph Approaches

3.1.1 Overview

At its core, extractive summarization is a method of choosing a subset of sentences within a document. Ideally this subset will maximize the amount of actionable information within the document while minimizing unnecessary information. In this report we focus on graph-based solutions to this subset selection problem; however, we clarify that they are not the only solutions. Optimization based approaches such as submodular optimization [20] [21] [9] and semantic volume maximization [48] are viable options. We refer to the cited literature for more information on those approaches.

In order to choose which sentences are best to include, we define a notion of centrality. In graph theory and network analysis, centrality identifies the most important vertices. Depending on the problem domain, centrality can be defined by a number of metrics. For example, in a graph representing a social network, where vertices are users and edges are created when one user follows another, centrality may be defined as the users with the most followers. In summarization, centrality is defined as the sentences which are most similar to the document as a whole.

This is a fairly nebulous definition of centrality, offering no insight into how to compute it. In fact, most graph summarization algorithms are simply different methods for computing sentence centrality. In many cases, sentence centrality is reduced to the centrality of their component words. However, this requires a defined method for computing individual word centrality.

One early method defined word centrality by the distance of each word from the centroid of the document in vector space [38]. The centroid of a document are words which have scores above a predefined threshold, determined by term frequency-inverse document frequency (TF-IDF), defined as the number of times a word appears in the document normalized by the number of times that word appears over a large, similar-domain document set [15]. The sentences with the most words within the centroid are closest to the centroid, and thus most central.

This strategy, and other word-based reductions, ignores the fact that sentences within a document are related to one another. In order to account for sentence level relationships, many algorithms take inspiration from Google's PageRank [35], which measures web page importance in order to be able to rank them for search engine results. First, a graph is built out of the webpages, where each vertex is a webpage, and a directed edge is placed between vertices x,y if x has a link to y. Then, each edge is given a weight, determined by the importance of source vertex x; a link is worth more if it comes from an important webpage. Initial importance is standardized across webpages and is updated iteratively to reflect the true importance of the webpage given the existing links between them. At equilibrium, webpages are ranked by the amount of weight directed towards them.

This graph-based approach is adapted to generic text ranking and applied to summarization by a number of algorithms. These approaches greatly improved the state of the art at the time of their inception. Two

of the most successful approaches are TextRank [27] and LexRank [11]. TextRank is an almost one-for-one implementation of PageRank applied to text domains. LexRank further provides improved definitions for similarity between words and sentences, as well as a minimum similarity threshold to create a graph edge. While we define the two similarity metrics used in these algorithms in the following section, it is important to note that other similarity measures may be used within the graph construction and may provide different performance depending on the particular text domain and dataset.

3.1.2 LexRank and TextRank

The graph-based extractive summarization models generally follow the same three-step structure. First, compute sentence similarity for each sentence pair within the document. Then, generate a graph by creating a vertex for each sentence and a weighted edge for each computed pairwise similarity. Finally, iterate over the weighted edges until an equilibrium is reached, and extract the sentences with the most weight directed towards them. The primary difference found across these algorithms is the strategy used to compute sentence similarity and generate the graph's edges.

In order to determine sentence similarity, the raw sentence text must be transformed into a new representation suitable for computation. One standard text representation adopted early into these sentence ranking strategies is the bag of words model. A bag of words of a sentence is the multiset of its words: a vector which maps word index to the frequency of that word in the sentence. Thus, each sentence becomes a 1-dimensional vector of the given vocabulary length. For use cases that involve summarizing a collection of documents, the vocabulary can be generated from that collection.

For TextRank, it is possible to compute the similarity between two sentences directly from this bag of words representation. The similarity score between any two bags of words is simply the number of words they have in common, normalized by length. Formally, given sentences S_1, S_2, this is defined as:

$$\frac{|w_k | w_k \in S_1 \& w_k \in S_2|}{log(|S_1|) + log(|S_2|)}$$

While this similarity metric offers the advantages of being simple and easy to compute, it does have at least two major weaknesses. It does not consider extraneous words that are in one sentence and not in the other. Words can only contribute positively to the similarity score, or contribute nothing at all. If a sentence has lots of extraneous words that do not appear in other sentences, signifying that it is dissimilar to the rest of the document, it is not penalized. Words that might indicate a sentence is not central are ignored. Furthermore, long sentences are unfairly rewarded, as they have more chances to match words against other sentences. Penalizing extraneous words could help both avoid sentences that are erroneously considered central and help prefer more succinct sentences.

The second issue in TextRank is that it does not consider that words have different natural frequencies or rarities. For example, the articles 'a', 'an', or 'the' are extremely common in the English language. The likelihood any one of these articles appears one or more times in a sentence is high, yet they contribute very little to the overall meaning of a sentence. A word like 'cabotage' is much rarer; it is expected to appear infrequently in sentences, and likely contributes significantly to the meaning of a sentence. It is reasonable to want to reward a 'cabotage' match between sentences more than an article match when determining the similarity between two sentences.

A standard method for incorporating word rarity within a collection of documents into the bag of words model is to replace the baseline word frequency with a Term Frequency–Inverse Document Frequency (TF-IDF) value. For a given word (term) t in a sentence, its TF-IDF score is its frequency within the sentence normalized by its frequency across documents. This scaling term is computed as $\log(\frac{N}{DF_t})$, where N is the number of documents in the collection and DF_t is the number of documents t appears in. This ratio is a rarity score; the fewer documents a term t appears in – the rarer it is – the larger the ratio. This new data encoding is the basis for the LexRank algorithm.

Though LexRank resolves the word rarity issue, it presents a new problem. Unlike TextRank, which can be used out of the box with no additional data whatsoever, a collection of documents is required to build the IDF counts. Furthermore, this collection must match the domain as the documents being summarized, since word frequencies change across domains. For example, words such as 'affidavit' and 'misdemeanor' may be relatively common in a corpus of legal documents, but they may not appear at all in religious text. In use

cases where there is a very large corpus of documents to summarize, it is possible to train the IDF counts directly. However, additional data may be required to adequately model the domain.

In addition to the adoption of TF-IDF, the other major difference between LexRank and TextRank is the similarity metric used. Rather than sum the number of matches, LexRank computes the cosine similarity. TF-IDF modified cosine similarity, the angle between two TF-IDF vectors, is defined as

$$cos(x,y) = \frac{\sum_{w \in x,y} tf_{w,x} tf_{w,y} (idf_w)^2}{\sqrt{\sum_{x_i \in x} (tf_{x_i x} idf_{x_i})^2} \sqrt{\sum_{y_i \in y} (tf_{y_i y} idf_{y_i})^2}}$$

By using cosine similarity, LexRank ignores the magnitude of the sentence vectors. This is desirable when working with input of various lengths – a term t occurring once in sentence x and twice in sentence y does not necessarily mean that y is more related to t than x: it might just mean that y is twice as long. Thus, cosine similarity normalizes for length differences between sentences implicitly instead of TextRank's explicit normalization.

Once the similarity for all pairs of sentences is computed, a graph can be generated. Each sentence is represented as a vertex in the graph, and an undirected weighted edge between each vertex is created with weight equal to the similarity score between the two sentences. This creates a complete graph; there is an edge between every vertex, regardless of how similar the sentences are. In algorithms like PageRank, an edge is created by the existence of hyperlinks, which either exist or do not; thus the generated graph misses edges between vertices that are not related. To replicate this behavior, the LexRank algorithm includes a pruning threshold for edges at graph initialization. If the weight of an edge falls below this threshold, it is removed from the graph. Thus, only significant similarities are included in the edges of the LexRank graph.

Once the graph is generated, weights are iterated over to determine which sentences are most central (have the most weight). There are a number of definitions that can be used to determine centrality. Degree centrality is most simple: it counts the number of edges pointing to a vertex, ignoring the edge weights. In complete graphs, such as the one generated by TextRank, degree centrality fails since each vertex has the same degree. A more common definition is prestige centrality, originally found in the PageRank algorithm. Prestige centrality uses the weighted edges to compute vertex centrality, then uses the vertex centrality to re-compute edge weights. This process is repeated until an equilibrium is reached, and neither vertex centrality or edge weights change.

For further details and exact implementations of the various LexRank and TextRank components, we refer to their original publications [11] [27].

3.1.3 Why Use Graphs?

Graph-based approaches offer multiple advantages. The most useful property of these algorithms is their unsupervised nature, requiring no labeled data to train. This is in stark contrast to recent neural network summarization systems, which require large quantities of domain-specific article-summary pairs. This labeled data, while plentiful in domains such as news articles, can be scarce to non-existent in others. Extractive graph algorithms provide a functional baseline for problem domains that do not have access to the training resources required for newer models.

Another limiting factor in model choice is access to appropriate hardware. Neural models can be computationally expensive, with the largest models often requiring multiple high-end GPUs running for weeks at a time to train. While running inference with an already trained neural model can be done on more limited hardware, runtime can still be an issue. Graph algorithms avoid these resource costs entirely. The training procedure, generating word and document counts within a corpus, is linear time with respect to the number of words, and is not dependent on specialized hardware. Inference is similarly fast and hardware-agnostic.

Outside cost analysis and resource requirements of graph approaches, extractive methods as a whole have a number of behavioral properties that are desirable. Due to the nature of extractive systems, output sentences will always be the same quality as the input article; if the input sentences are grammatical and fluent, the output sentences will be grammatical and fluent. Fail cases are limited to poor choice of sentences to extract and do not affect the grammar or fluency of the sentences. Catastrophic errors, such as nonsensical sentences, are avoided. Furthermore, since sentence similarity is directly related to word matches, short sentences will generally not be selected for output as they have fewer words to match. In most cases, short

sentences will not contain as much meaningful or relevant content, or they may be more reliant on nearby context. Thus, they should usually be excluded from the summary.

Finally, many open-source implementations of graph algorithms exist that offer both easy to use interfaces and generic pretrained models. Combining the ease of use with high performance floors and low resource costs, extractive graph summarization systems remain a strong solution for many use-cases in the modern day.

3.1.4 Why Avoid Graphs?

Despite their desirable properties, extractive graph approaches do have a glaring weakness: a low performance ceiling. This can be attributed to a number of problematic underlying assumptions. The first faulty assumption is that the collection of most central sentences provide a complete summary. Due to the nature of centrality, sentences that are similar to each other are highly rewarded; the most central sentences might all be variations of the same idea or topic. If a document contains more than one idea or topic, a good summary ideally covers all of them, which is not necessarily true in a centrality system. There is a disconnect between which sentences are best to include in a summary and which sentences are most central to the document.

There are graph rank variants which use diversity in addition to centrality to determine rank in graph algorithms. Grasshopper [52] incorporates absorbing Markov chain random walks and DivRank [26] adds in reinforced random walks. We refer to the original papers for details on how random walk variations result in increased diversity. For document corpora where a primary concern is diversity of extracted information, it is possible that these variants adequately alleviate concerns of information duplication in the summary.

One of the most egregious assumptions made by graph systems is that sentence similarity can be accurately modeled by exactly matching words in a bag-of-words vector, without real underlying language knowledge. There are many relationships that this does not model, such as synonyms, antonyms, and part-to-whole (e.g. toe to foot). Since relationships like synonyms exist and sentence structure is not fixed in language, it is possible for two sentences to be similar without sharing a word. For example, consider the sentences 'Math is fun for children.' and 'Kids really love mathematics.'; neither of these sentences share a word, but clearly share similar sentiment, subject, and object. Since no words are exactly matched here, these sentences would erroneously not be classified as similar. Thus, even if the most central sentences are guaranteed to be the best possible extractive summary, without a mechanism for modeling language and capturing complex relationships between words and sentences, it is not possible to perfectly determine the most central sentences.

There is one generic faulty assumption made by all extractive systems: that sentences from the source document can be reasonably extracted into a summary. Extracted sentences are not written with summarization in mind. They may contain extraneous information that bloat the summary, or they might be dependent on nearby context that does not get extracted. Even if the sentence is self-contained, human-generated summary sentences will be written to condense information into as few words as possible, which will not be generally true of sentences found in the source document. Finally, while sentences will be at least as fluent and grammatical as the input, the overall summary will be disjointed. The summary will likely be composed of sentences that were not originally written next to each other and do not flow into each other. Sentences can appear in a different order in the summary than they do in the original text, making causal or logical relationships between statements difficult to discern. This can cause the summary to be difficult to read as a whole.

Together, these assumptions cause graph-based extractive methods to fall behind modern abstractive approaches in terms of summary quality. Furthermore, it is impossible for extractive systems to get high automatic scores such as ROUGE; the gold label summaries will contain novel words and sentences not seen in the document. Even if the extractive systems always choose the optimal set of sentences to maximize ROUGE score, it will still be below a ceiling determined by the different words used in the the gold summary and the original text. Without being able to rephrase and generate new language, extractive systems cannot find and output the novel sentences in the summary. In cases where the best performance is a priority, graph methods, and possibly even extractive methods as a whole, may not be the appropriate choice.

3.2 Extractive Neural Networks

3.2.1 Graph Networks for Extractive Summarization

One solution for the lack of language understanding in classical graph approaches is the incorporation of neural networks. Neural networks have shown success in language modeling tasks [14]. It is possible by leveraging neural language capabilities and incorporating them into the graph algorithm, complex relationships between words and sentences can be captured.

Instead of using a bag of words representation for the sentence, a sentence can be represented as a sequence of trained word embeddings such as word2vec [28]. These word embeddings can map similar words close together in vector space and are a natural representation for computing similarities. Alternatively, rather than use a vector of word embeddings, a single sentence embedding can be computed by a neural network. In these representations, computing similarity can account for word meaning, word order, word relationships, and create a more accurate representation for graph creation.

Incorporation of neural networks into the graph algorithm is not limited to the input representation. As part of the Machine Intelligence for Language Understanding (MILU) project, we experimented with replacing both the bag of words representation as well as the similarity computation for graph-based extractive summarization. While cosine similarity or other vector distance metrics can be viable, they are intended as general purpose distances and are not specific to the text domain. We hypothesize that trained specialized networks, specifically BERT sentence embeddings [10] for the representation and a Siamese network for the similarity metric [6], can better model the similarity between two sentences, resulting in more accurate centrality predictions.

In our experiments the two neural network replacements, either separate or combined, did not result in better performance in terms of ROUGE score. The sentence embeddings failure may have been caused by a poor choice in embedding; newer research shows that the default BERT sentence embeddings do not work well for sentence similarity tasks [40]. It is possible different sentence embeddings, such as SentBERT [40] or sent2vec [36], would be a more successful choice. Alternatively, the embedding replacement may have failed because the similarity metrics used were not capable of modeling the additional language information or were not compatible with vector embeddings. For example, it is not guaranteed that two similar sentences are mapped nearby each other in vector space, as determined by Euclidean or cosine distance.

There were a number of difficulties with training a Siamese network. The largest issue is the lack of available paired labeled data. While there exist text entailment datasets, such as SNLI [4], which include pairwise sentence classifications, these are generally not real valued judgements. Rather, these datasets classify sentence pairs as supporting, neutral, or contradict. This can be re-interpreted as similar, neutral, not-similar; however, this classification system is not cleanly ported into graphs. Graphs expect a real-valued similarity between zero and one. Even in binary classification, where the neutral tag is dropped, a score of 0.6 does not mean 60 percent similar, but rather a measure of how confident the model is that the sentences are similar.

Replacing the classification labels with a real value proves unsuccessful as well. It is not clear how to obtain human judgements of real similarity values; it requires the annotator to answer what it means to be exactly some amount similar. Combined with the expense of generating human annotations, manual labeling of data is not a feasible option in a majority of cases. Furthermore, automatic labeling provides little to no benefit. In order to automatically label sentences, a computable similarity metric is required. The Siamese network trained on this data could only ever be as good as the similarity metric used in the data creation—at which point, it is better to use that metric in the graph directly.

Our conclusion is that it does not seem immediately possible to replace graph based algorithms piecemeal with neural networks and expect better results. This does not rule out top-down redesigns with neural networks in mind. Full incorporation poses a number of new challenges, primary among them the need for additional labeled data. However, additional labeled data allows for more complex end-to-end neural architectures that generally outperform the graph algorithms.

3.2.2 End-to-End Networks for Extractive Summarization

As labeled datasets have gotten larger and neural network language capabilities have improved, an increasing number of end-to-end neural summarization systems have appeared. We detail two particular models of note

which have proven results, but specify that these two chosen models are representative of particular structures with many available options.

The first neural model is the SummaRuNNer [30], an attentional recurrent neural network (RNN). This model treats summarization as a binary sequence classification problem, where each input sentence in the document is classified either as part of the summary or not. One problem with treating summarization as a sequence classification task is that no such labeled data exists. Summarization datasets contain human generated summaries, which are always abstractive in nature. In order to train these models, SummaRuNNer proposes a method for converting the existing abstractive datasets into extractive datasets. Essentially, the conversion follows directly from the idea that the extracted sentences should maximize the ROUGE score with the existing abstractive summary. Because it is practically impossible to try every possible combination of sentences in the document to find the optimal set of sentences, SummaRuNNer applies greedy search to quickly find a decent approximation. The set of sentences found are then labeled as part of the summary, and all other sentences are labeled as not.

Once the data is converted, it is possible to use any sequence classification neural network. One caveat is that the model is performing document summarization; it is important that the model read the whole document at once and perform classification of each sentence with respect to the other sentences, instead of one at a time. In order to be able to model this dependency, SummaRuNNer uses a standard neural architecture: an encoder RNN, decoder RNN, and an attention layer [1] to transition between them. This data conversion and neural architecture provides state of the art extractive performance and shows that summarization does not need unique architectures to be successful.

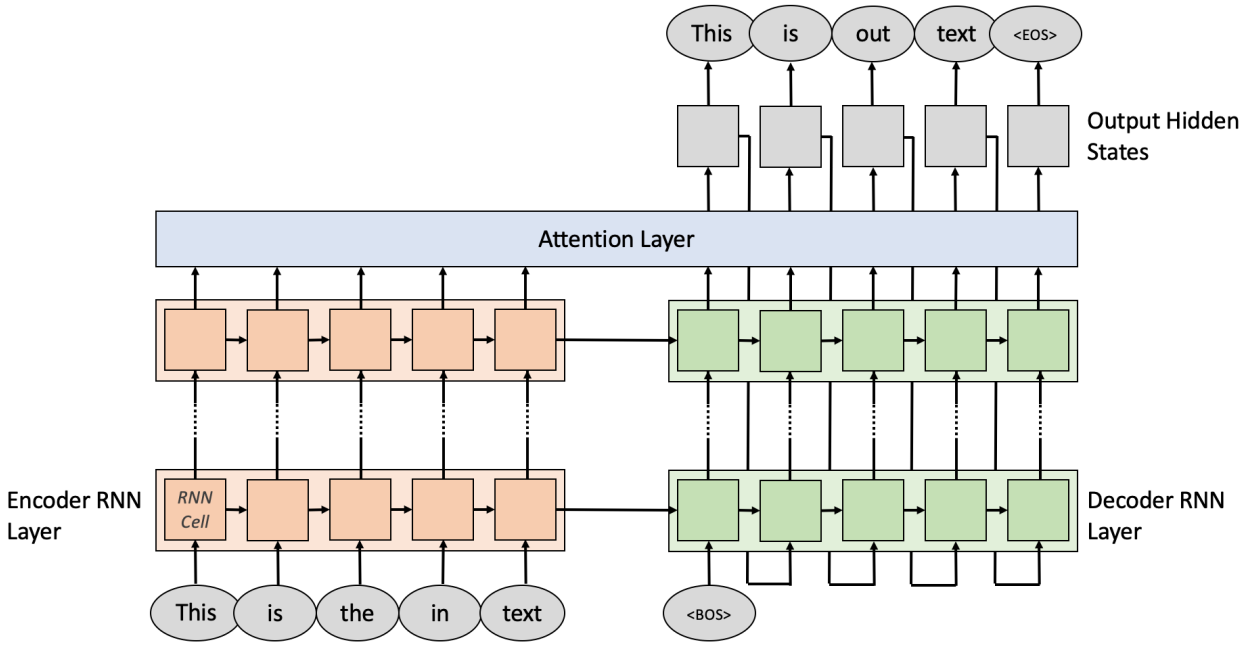

Figure 1: Encoder-Decoder RNN With an Intermediate Attention Layer

The second model we detail here is Cheng and Lapata's hierarchical pointer model [5]. Like SummaRuNNer, Cheng and Lapata first convert abstractive summarization datasets into extractive datasets. However, rather than treat the task as sequence classification, they simply replace the gold-standard abstractive summary with the approximately optimal extracted sentences. While they do not use greedy search and ROUGE score to determine this approximate optimum, it follows the same idea, and we refer to the original paper for implementation.

Cheng and Lapata's contribution is in the model architecture choice. Rather than use an RNN, they opt for a hierarchical encoder combined with an attentional decoder. The attentional decoder is used like a pointer network [44]. Usually, attention is an intermediate step used to connect the encoder hidden states to the decoder, as seen in SummaRuNNer. In pointer networks, the attention is used to directly select the

sentences or sequences in the input to use as output. Essentially, the attention points to the sections of the input to extract, avoiding the need for an additional decoder.

Together, these models show two possible approaches to end-to-end neural extractive summarization. However, while many models fall into these structures, they are by no means the only neural extractive solutions. Surveying all successful implementations is outside the scope of this report, so we limit the discussion to a few structures we believe to represent a large portion of available and successful solutions.

4 Abstractive Summarization

4.1 Recurrent Neural Networks

Before the advancements in neural networks in 2014, abstractive summarization was very rare. However, the improved language modeling and generative capabilities of modern neural networks began to allow for more abstractive solutions, where entirely new text is generated to create summaries. The earlier abstractive networks [41] [1] were generally some form of encoder followed by a generative language model. This encoder could be something as simple as the bag of words seen in graphs, or more complicated models like convolutional networks or attentional RNNs. This model structure is applicable to any sequence-to-sequence task, as the language model can be trained on generic input sequence to generic output sequence.

Attentional encoder-decoder models, similar to SummaRuNNer, began to see significant success after their generative capabilities were proven in similar sequence-to-sequence tasks such as machine translation [1]. Instead of using this model structure as a sequence classifier as in SummaRuNNer, abstractive variants are trained using a generative decoder. The model is trained directly on sequence-to-sequence data, where the encoder learns to encode the input sequence (document), and the decoder learns to generate an output sequence (summary) from that encoding. These abstractive attentional encoder-decoders found immediate success, reaching state-of-the-art results [29].

Despite their strength, attentional RNNs still produce a number of undesirable effects that often make individual summaries significantly worse. They can produce factually incorrect information, as they generate the information from scratch rather than extracting the information directly in a correct form. Models also had a tendency to repeat themselves and could get stuck in generative loops. Finally, since generative decoders have to create a probability distribution over words, generation is limited to a small vocabulary of words. This makes it impossible to generate rare words that are not contained within the vocabulary. In some cases, these problems could be so extreme that a generation summary contains no information or is otherwise unreadable.

A number of models address issues while staying within the attentional RNN structure. Zeng et al. (2016) [49] add a copy mechanism to the attentional RNN, allowing it to directly copy out-of-vocabulary words. See et al. (2017) [42] add in a coverage mechanism that tracks what has already been attended to and discourages further attention being applied to those sections, reducing the amount of repetition in the output. See et al. (2017) further try to solve the generation of factually inaccurate information by adding in an extractive pointer network. The pointer network is combined with the generative RNN to be able to directly copy words from the document in addition to generating new text, allowing it to extract factual information directly and generate new text to piece those facts together.

While these extractive and abstractive attentional encoder-decoder RNNs have been surpassed in terms of raw performance by the newer Transformer model detailed in the next section, there are still reasons to choose these systems. Older systems, such as extractive graph systems, offer a high performance floor but low ceiling for little to no data requirements and computational cost. Attentional encoder-decoder RNNs offer better overall performance, though with the potential of catastrophic failure on individual examples, at higher resource costs and a labeled data requirement. The current state-of-the-art Transformer model provides the best possible performance, but at the highest resource and computational cost. Thus, RNNs essentially split the difference, providing neither the lowest resource cost nor the highest performance, but are somewhere in the middle for each. For certain computational environments and problem spaces, this may be desirable.

4.2 Pretrained Transformers

The current state of the art in many natural language processing tasks are the recently developed Transformer models [43]. The Transformer model consists of sequential attention blocks, which are themselves series of attention layers and feedforward networks. For specifics on these transformer blocks, see Vaswani et al. (2017) [43] for the original model description. While Vaswani et al. (2017) use two Transformers for their translation task, an encoder and decoder combined, it is common in subsequent work to see only the encoder transformer, attached to a small, linear output layer [10].

In figure 2, we depict how an attention block is formed, either as an encoder layer or a decoder layer. Both versions include a self-attention layer which attends directly to its input text. The decoder additionally has a cross-attention layer, which attends to the output of the self-attention of the decoder and the output of the self attention in the encoder. Despite a lack of cross-attention, many single-transformer models, which input the article and summary together into a single transformer, are called decoder-only models. The model combines encoding and decoding into a single transformer and uses only self-attention. In order to avoid any confusion on whether or not a model includes cross-attention, we use the terminology single-transformer for decoder-only models without cross-attention, and two-transformer for encoder-decoder models with cross-attention.

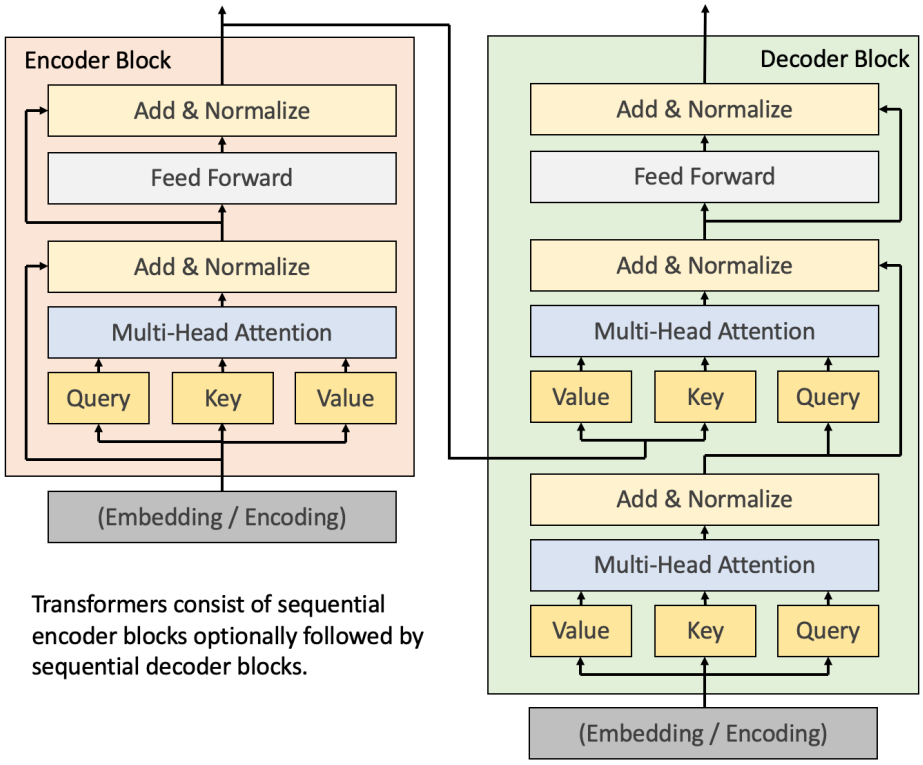

Figure 2: Encoder and Decoder Transformer Block in Vaswani et. al (2017)

Unlike its predecessor recurrent neural networks, every layer in the Transformer model is highly parallelizable. This has allowed for rapid growth in the number of parameters in the model and in the amount of data required to train on. The largest Transformers often require an array of dozens of GPUs and weeks of compute time to properly train. Unfortunately, this kind of compute power can be prohibitively expensive for users and prevents personal end-to-end training.

To combat this issue, Transformer training is split into two stages. The first stage trains the model on generic language modeling tasks, without an attached output layer or decoder. This incorporates the large quantity of existing unlabeled text data, allowing the language understanding portion of the model to be trained on more than the relatively small amount of labeled data. More importantly however, it decouples the language understanding portion of the model from the target task. Thus, this training step can now

transfer across multiple models or tasks. A number of first-stage Transformer models, such as BERT [10], GPT2 [39], or RoBERTa [24] are publicly available through Huggingface's open-source library 'Transformers' [1], which is available for both PyTorch [2] and Tensorflow [3], two of the most popular deep learning frameworks. These various models are all trained on various unlabeled corpora with different training objectives, making it highly likely that there exists a pretrained model suitable for any given language task or domain.

By performing the majority of the training in this language modeling pretraining step, Transformer models have become widely available for use in downstream tasks. The second training step 'fine-tunes' the model to the labeled data specific to the task, such as summarization. It is at this point that a traditionally small, task-specific decoder or output layer is introduced and trained from scratch. Since the rest of the model is already mostly trained, and the decoder can be small, this step can be completed in reasonable time. For example, fine-tuning on a dataset with 300K training pairs might take a few days on a single GPU, rather than weeks on dozens of GPUs.

Thus, the largest, most powerful language models have become easily available for use in summarization. The current state-of-the-art summarization systems all leverage the Transformer models and their powerful language modeling in order to generate extremely high quality, fully abstractive summaries.

4.3 Evaluation of Transformer-Based Abstractive Summarization

4.3.1 Methodology

We experimented with a number of different Transformer models in order to best understand how to use them for summarization and which Transformer model was best suited for it. We quickly focused our efforts onto three choices: BERT [10], GPT2 [39], and a custom Transformer model. These three models were chosen to explore the various effects of pretraining. The GPT2 model is pretrained with the standard language model task: predict the next token given the current input. BERT, in order to implement bi-directional encoding and sentence level understanding, used two auxiliary tasks: masked language model, and next sentence classification. The masked language model task is similar to next token prediction—however, rather than predict the next token, it instead masks a small portion of the input and then predicts those masked tokens. BERT is also pretrained on next sentence classification, a training objective which classifies whether a sentence follows another one. Finally, as a baseline, the custom Transformer is not pretrained at all.

These three models additionally diverged in model structure. The BERT model is a combination of two Transformers: A pretrained encoder Transformer (BERT) which encodes the input document, and a decoder Transformer which takes the encoding and performs generation. The custom model shares this structure to facilitate a direct comparison of the value of pretraining without considering major architectural changes in the model. GPT2 on the other hand is a single Transformer model which does both the encoding and generation combined using a simple linear output layer. In the single Transformer case, rather than generating a summary based on an encoded version of the input, the summary is treated as the next text to be generated after the input text, like a TL;DR statement at the end of a long article.

Due to the prohibitive cost of pretraining models, we do not pretrain the BERT and GPT2 models ourselves. Instead, we use off-the-shelf pretrained models provided by Huggingface's Transformers library. Specifically, we use their 'bert-base-uncased' and 'gpt2' models. The 'bert-base-uncased' model is 12 layers, 110 million parameters large and is trained only on lower case English data. The 'gpt2' model is 12 layers, 117 million parameters large and includes cased English in its training data.

We train our three models on two publicly available news summarization datasets: CNN-DailyMail (CNN-DM) [31] and eXtreme Summarization (XSum) [32]. CNN-DM is a combination of news articles from the CNN and DailyMail websites. The summaries for these articles are human generated: each article is accompanied by highlight sentences used to describe the article in quick detail. These highlights are used as reference summaries. XSum was generated by another means: articles from BBC were downloaded and the titles of the articles were used as summaries. One of the major differences between the two datasets is the length of the reference summary. XSum references are extremely short, almost never more than one sentence; CNN-DM are on average two and a half times longer, as they are multiple sentences. These datasets provide

[1] https://huggingface.co/transformers
[2] https://pytorch.org
[3] https://www.tensorflow.org

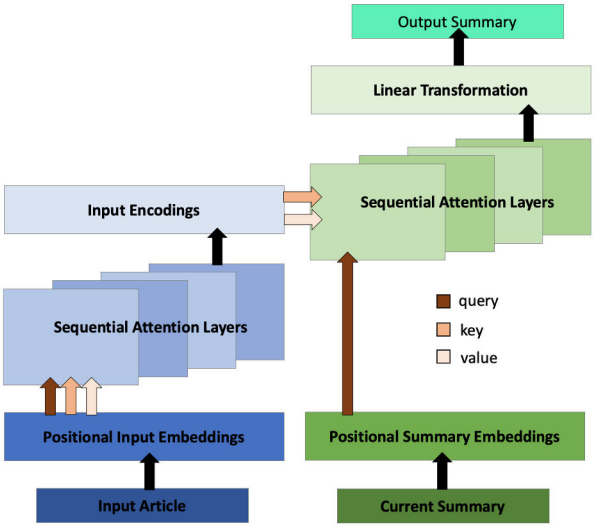

Figure 3: Two Transformer Model

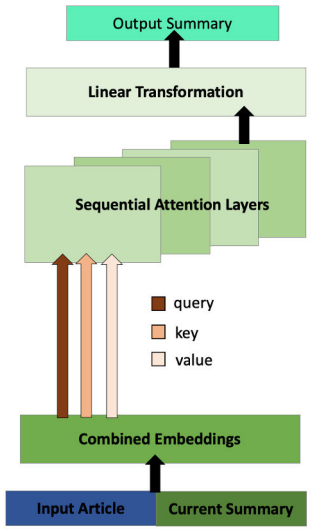

Figure 4: One Transformer Model

two unique challenges: one rewards summarizing to the minimum possible amount (XSum), whereas another rewards summarizing the major points of an article (CNN-DM).

Thus, our experiments test three objectives. We explore the benefits of pretraining, as well as variations in pretraining objectives. We compare the two-transformer approach, where encoding and decoding are delineated into separate portions of the model, to the single transformer case. We test these models on two variations of the summarization goal in order to determine if there is a clear best model, or if it depends on the particularities of the summarization data and goal.

4.3.2 Results

Below, we report the scores we obtained for the custom, BERT, and GPT2 models on the XSum and CNN-DM datasets.

Model	R1	R2	RL	Avg. Len.
Custom	29.32	9.61	23.47	18.36
BERT	34.50	**13.33**	**28.08**	17.42
GPT2	**34.80**	13.00	27.39	**17.33**

Table 1: XSum ROUGE Scores for Custom, BERT, GPT2 Transformers

Model	R1	R2	RL	Avg. Len.
Custom	38.37	16.46	35.20	45.09
BERT	37.62	17.00	34.78	**37.13**
GPT2	**40.58**	**18.06**	**37.61**	54.62

Table 2: CNN-DM ROUGE Scores for Custom, BERT, GPT2 Transformers

The most immediate result found in these models is the high benefit of using pretraining for XSum. Table 1 shows that while the custom model was able to find some success on the task, ROUGE score on the output was significantly worse than the pretrained models. On the other hand, table 2 shows that for CNN-DM, this effect is diminished: the custom model was able to outperform the BERT model despite the lack of pretraining, though both models were noticeably worse than GPT2. One reason why BERT in particular

would do so poorly is a mismatch in pretraining objective and downstream task. Where GPT2 is pretrained on documents, BERT is pretrained on sentences. Thus, BERT will be biased to generate the end-of-sequence tag after every sentence, ending the summary early.

This bias is supported by the average length of the XSum and CNN-DM output. Both BERT and GPT2 have almost identical output lengths when the output is expected to be exactly one sentence long (XSum). Here, having additional bias of generating the end-of-sequence tag after completing a sentence would not negatively affect generation. However, when output is expected to be between two and three sentences (CNN-DM), this bias can frequently lead to ending generation too early. This effect is seen exactly in the CNN-DM results: BERT output is on average 17 tokens less than GPT2 output on CNN-DM, roughly a full sentence length difference. This causes such a negative affect on the score that it is actually better not to use a pretrained model at all over the BERT model. It is clear from these results that not only is it important to pretrain the Transformer model, it is important to pick a model whose pretraining schema is similar the target downstream task.

While it is likely that GPT2's increased performance over BERT can be attributed to the differences in pretraining between the models, it is also possible that the decoder-only model structure is simply a stronger choice. Unfortunately, during the time of these experiments, Huggingface's Transformer library, which provided the pretrained models, had a far more limited selection of models. For example, until very recently, this library did not support using a GPT2 pretrained model in an two-transformer structure—it was only available as a single-transformer model. Future work in this area would include an investigation into how a two-transformer GPT2 based model would perform against the two-transformer BERT model and single-transformer GPT2 model, to determine which of the two possibilities it is.

There are reasons to believe that the decoder only model is in fact the better structure. This is largely due to the fact that it takes better advantage of pretraining. Pretrained models are only available as single Transformers—decoders are task specific and cannot be trained in advance in general. Thus, the two-transformer BERT model would have instantiated a large decoder Transformer with no pretraining. As evidenced by the custom model in XSum, pretraining can be highly valuable, and instantiating a large fresh decoder diminishes the total effect pretraining has on the model. Furthermore, unlike the two-transformer structure, which condenses the input document into a single encoding, the single Transformer allows for generated tokens to directly attend to input tokens. It is possible that this ability to directly attend allows for better context management and language choice in output and could be the reason that GPT2 is the best scoring model on both datasets.

This direct attention between generated tokens and input document tokens is not without its own costs. The attention mechanism in Transformer models is quadratic with respect to number of tokens it must attend to – every token has to attend to every other token. Thus, the runtime and resource cost of using a single-transformer structure is significantly higher than a two-transformer structure, as the latter splits the document and generated tokens into two separate attention vectors, reducing the quadratic cost increase. For short inputs, its unlikely that these costs will come into play – but as documents get longer, it becomes increasingly more difficult to manage a decoder-only model.

4.3.3 Lessons Learned

Throughout our experimentation, we came across a number of tips and tricks we felt were important to detail for future developers considering a Transformer model. For training, finetuning should be fast in general. While, depending on dataset size, it can still take quite a few days, the model should be given to converge rather quickly, usually within just a couple of epochs. Longer training is generally, but not always, indicative that something in the training procedure is not working as expected.

Another important tip for training a Transformer is to use a large batch size. Due to the immense size of these models, they need significant amounts of data to train. It is important that when the model updates its parameters during training that the update is supported by a large number of examples. In cases where the model is very large or the inputs themselves are very large, such as full documents in summarization, it is unlikely that a large batch size will fit onto a smaller GPU. If both the model and the data input are very large, it is possible even larger GPUs will not be able to contain everything. It is recommended to use gradient accumulation across batches in order to emulate larger batch sizes. In our experience, an accumulated batch size of around 64 was the minimum we would use.

In our experience performing text generation during evaluation, models had a tendency to generate the end-of-sequence tag, signaling that generation is over, far too early. This results in output that looks only half complete. To avoid this issue, aggressive normalization of end-of-sequence tag generation is required. Example normalizations include requiring a minimum number of tokens generated before end-of-sentence, or smoothing the probability of end-of-sequence by current generation length. If the opposite problem is occurring, where end-of-sequence is not being generated enough, one possible solution is to use beam search decoding with large beam sizes. Larger beam sizes offer more opportunities to generate the end-of-sequence tag at any given step and may help alleviate the issue.

Finally, the approaches used for summary generation with Transformers are mostly the same as earlier neural networks, such as RNNs. They use the same decoding algorithms to determine what token to generate next given a probability distribution generated by the model, such as nucleus decoding [13], greedy search, or beam search. Improvements to these algorithms that work on early neural networks have a high likelihood to work for Transformers. For example, length normalization and trigram filtering, common normalizations in beam search with RNNs, work for Transformers. We would recommend using open source implementations of these decoding algorithms, specifically Huggingface's Transformers generation code, which was recently improved in their 3.0 release of the library.

4.3.4 Summary Coverage

Following the finding that improvements from decoding algorithms carried over from RNNs to Transformers, we experimented with using an attention coverage mechanism that has found moderate success in RNNs [45] [18]. Attention coverage adds a reward term into a generation candidate's score, computed from that candidate's attention vector. The idea is twofold. First, the attention vector should be trying to attend to everything in the original document: the more words that were attended to, the better our summary 'covers' the document. It is not essential that something is generated for everything in the input – only that the attention vector attended to it. This essentially penalizes low valleys of attention across the candidate, where sections of the input document are ignored.

The second component to attention coverage serves as the actual reward. At each generation step, we reward attention vectors which have high peaks. Essentially, when generating the next word, it is important that the word is focusing on something specific in the article. It needs a direct reason for being generated. This avoids the case where the output starts becoming too generic. Combined with the penalty for low valleys, this incentivizes the attention vector to maximize its attention across each word in the input for at least one step in generation. Thus, every part of the input document should be present in the output summary. Ideally, this will provide a significant increase to content accuracy.

Table 3: XSum No-Coverage vs. Coverage ROUGE Scores

Model	R1	R2	RL	Avg. Len.
Custom	29.32 / **29.68**	**9.61** / 9.54	23.47 / 23.45	**18.36** / 20.94
BERT	34.50 / **34.65**	**13.33** / 13.20	**28.08** / 27.85	**17.42** / 20.48
GPT2	34.80 / **35.16**	13.00 / 12.99	27.39 / 27.43	**17.33** / 18.81

Table 4: CNN-DM No-Coverage vs. Coverage ROUGE Scores

Model	R1	R2	RL	Avg. Len.
Custom	38.37 / **39.28**	16.46 / **16.69**	35.20 / **36.04**	**45.09** / 53.77
BERT	37.62 / **39.61**	17.00 / **17.60**	34.78 / **36.70**	**37.13** / 44.59
GPT2	40.58 / 40.53	18.06 / 17.99	37.61 / 37.55	54.62 / **51.39**

On the surface, the results are fairly promising. ROUGE score across model types mostly increased; for CNN-DM, every ROUGE metric increased, often by a full point or more. For XSum, the differences are a little finer. R2 and RL might increase or decrease, but often not by enough to eliminate random variance as the cause. R1 however, increased by nearly half a point in both the custom model and GPT2 model; the BERT model similarly saw an increase, though to a smaller degree. Not only did the ROUGE score increase,

Table 5: Human Evaluations of Custom and GPT2 Coverage

Model A	Model B	Raters	Articles	Fluency (A/B)	Content (A/B)
Custom	+Coverage	5	30	32.7 / 67.3	40.7 / 59.3
GPT2	+Coverage	3	40	40.8 / 59.2	45.8 / 54.2

human evaluators generally preferred the coverage output to the standard output in direct comparisons of two model types, determining that coverage output was more fluent and more content accurate around 60 percent of the time. Manual inspection of the output does verify that significant word choice changes occur when generating with or without coverage (all other normalizations standardized).

One observation that arose from the coverage experiments was that the coverage output was almost always longer. In order to verify that the score increase was directly tied to the word choice changes, and not the fact that it had more tokens, we tried generating coverage and non-coverage outputs of fixed lengths.

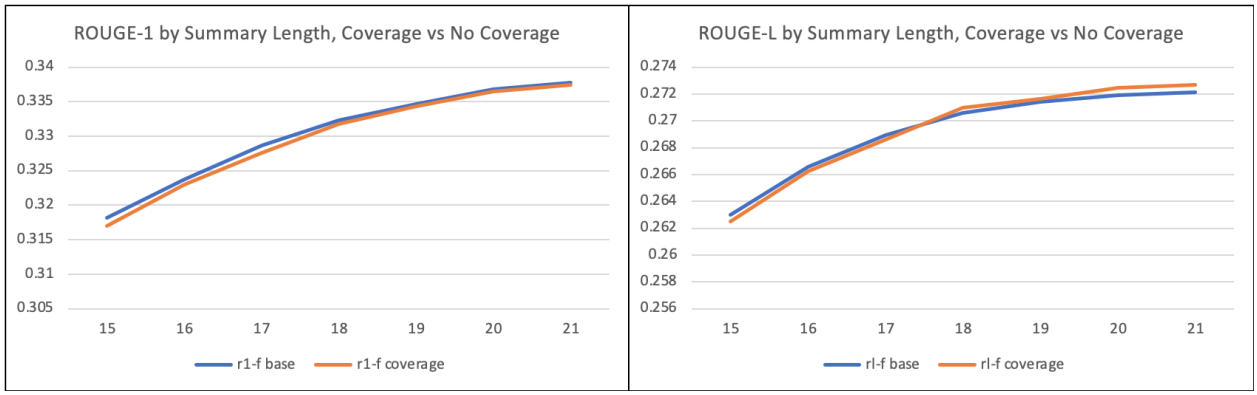

Figure 5: BERT XSum ROUGE Scores for Fixed Lengths

Figure 5 shows ROUGE-1 and ROUGE-L scores for fixed lengths for BERT models with and without coverage on XSum. The scores for the two models are almost entirely identical in both graph: at each possible output length, model score is within a tenth of a point. While not displayed, this trend is true for over rouge score variants such as ROUGE-2. Furthermore, though only the BERT XSum graph is displayed as it had the largest change in tokens, this result holds true for the other models. Immediately, it becomes obvious that word choice is almost irrelevant in the score increases seen; rather, length is the driving factor. Likely, this means that the output length is short enough that the recall computation in ROUGE is contributing more to score changes than precision – thus, as more words are generated, there are more opportunities to match words against the reference and increase recall, increasing the overall score even at the cost of lower precision.

It becomes clear that ROUGE score is rewarding lengthier outputs. This poses a significant problem, since the primary goal in summarization is to output the most accurate content in the smallest amount of text. This mismatch between the evaluation metric and the goal makes it difficult to trust ROUGE to determine the best model. Combined with the decorrelation with human judgement witnessed as model summarization quality increases, it is not clear ROUGE can be trusted to evaluate high end summarization at all.

Due to this finding, it is not obvious that the coverage mechanism provides a real benefit. It is somewhat promising that the human evaluations generally preferred the content of the coverage output. This is not necessarily a guarantee that the content was in fact improved but could be evidence that humans share the same bias towards longer output. This question is outside the scope of this project. For the time being, it remains unclear whether the coverage mechanism is simply adding an unwanted increase to length or whether it brings a legitimate content improvement.

4.4 Limitations of the Transformer

A number of limitations regarding Transformers make them inapplicable in certain problem spaces. The first is the models' inability to handle very long input. Unlike many non-neural systems, Transformers process the entire document at once in order to be able leverage their attention mechanism across sentences. Since attention is quadratic with respect to input size, as each token attends to every other token, there is both a resource requirement and dependency explosion as the input length increases. As such, the attention mechanism begins to fail once input grows too long. To avoid this issue, many pretrained Transformer models, including BERT and GPT2, incorporate a maximum input length. In the example models listed, the maximum lengths are 512 and 1024 tokens respectively. If the source document is longer than the model's maximum length, either a different model is required or the document must be truncated in some way.

There are a number of Transformer models which attempt to address this long input issue by modifying the attention mechanism. Two such publicly available models which we discuss are the Reformer [16] and Longformer [3], but other solutions exist such as the Performer [7]. The Reformer attempts to prevent quadratic explosion by hashing the input tokens in advance and only allowing tokens to attend to other similarly hashed tokens. By using multiple different hashes simultaneously, tokens can still attend to a large portion of the input while avoiding the quadratic increase. Longformer replaces the standard attention mechanism with a local windowed attention and global task attention, creating a linearly scaling attention mechanism. These two models are shown to handle longer inputs better on some example problems but it remains to be seen if this property is true in general.

Another major limitation with the Transformer model is its dependency on labeled data. While some summarization systems, such as graph summarizers, have been wholly unsupervised, Transformers require document and summary pairs to finetune on. Even with the split training step, which helps reduce the amount of labeled data needed, it is still a necessity. In many domains, the amount of labeled data available will be too small, or even be nonexistent, to finetune. It may be possible to transfer a model that was finetuned on a publicly available labeled set; for example, a news dataset like XSum. However, this provides its own sets of potential issues. In the XSum example, the structure of news articles usually places the most relevant information in the first few sentences. If the target documents do not follow that pattern, the trained model will unfairly weight the first few sentences, and relevant information in later portions of the document may be ignored.

Recent work in Transformer pretraining may address the transferability of Transformer models for summarization. Zhang et al. (2020) introduce a self-supervised summarization-based pretraining objective to the Transformer named PEGASUS [50]. In PEGASUS, important sentences in training documents are masked out, and are then generated as output sequences from the remaining input sentences, similar to an extractive summary. This pretraining objective effectively converts massive unlabeled corpora into a task extremely similar to abstractive summarization. Zhang et al. find that PEGASUS Transformers are very effective in low-resource summarization tasks, beating the state-of-the-art on six datasets with 1000 examples or less. By taking better advantage of pretraining data and objectives, it is possible that summarization in previously unseen domains is viable with a Transformer.

Finally, there still exist issues with Transformer models in general, separate from summarization. One such issue that might arise is a progressive drop in quality of generated text. Many of these models are very capable of generating one to three sentences with consistently high quality; however, as the output starts getting into its fourth sentence or further, that high quality is generally lost. As generation continues, the longer output might degenerate fully into unreadable nonsense. This follows directly from the long input issues seen in attention; as the length of the output increases, the number of tokens to attend to increases. Fortunately, the goal of many summarization systems is to produce as short an output as possible, so this can usually be avoided.

5 Multiple-Document Summarization

5.1 Overview

While the above discussion has focused on summarizing a single document at a time, another problem contained within the summarization umbrella is multiple-document summarization. Here, multiple documents

are considered together, and a single output is generated summarizing the collection as a whole. Currently, multiple-document summarization remains a mostly unsolved problem, with little to no success compared to single-document.

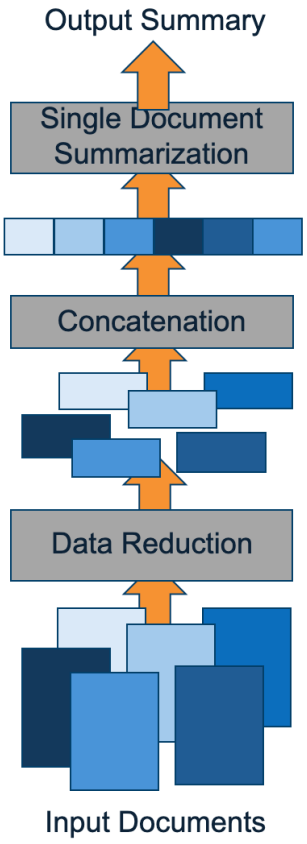

Figure 6: Two-Step Summarization

While this failure is largely due to the exacerbation of length and data issues presented in the previous section, there is also the question of how to formulate the multiple document summarization problem at all. Exactly how to encapsulate multiple documents together into one summary is not clear and is extremely context and goal dependent. For example, possible goals for summarizing a set of documents could be to extract shared facts across the documents, or describe the general themes of the documents, or simplify and condense information. All these variations would require different training objectives or different labeled data and may not be translatable to other multiple-document summarization use cases.

After deciding how to formulate the multiple-document summarization problem, there is still the issue of training data. In general, there are almost no multiple-document summarization datasets. Those that do exist are generally very small [34] [12], though there are at least a few exceptions to this; two exceptions are the WikiSum [22] and Multi-News [12] datasets. WikiSum uses Wikipedia articles as a summary, using the references and google results for the article title as the input documents. Multi-News consists of news articles and professionally-generated summaries by news editors from the site newser. Given the small number of training datasets, the likelihood that the target problem formulation and text domain together match one of these existing datasets is slim.

One potential solution to this data issue is to use unsupervised graph methods, which have been established to have a decent performance floor despite requiring no labeled data. In fact, most early multiple-document summarization systems have been variations of the PageRank algorithm. More recent work involves using graphs in tandem with an extractive neural models [46] [47]. Unfortunately, while extractive systems may have been well-suited towards single document summarization, they fall short in multiple-document scenarios. Extracted sentences offer poor substitutions for summaries in the single document case. This is exacerbated in the multiple document case. It is unlikely that any single sentence across a multiple-document example would serve, even poorly, as an output sentence that matches any of the use cases described earlier. As such, it is highly desirable to use abstractive methods for multiple-document summarization.

Assuming the problem formulation is established and labeled training data is found, there is still the issue of how to build a model to handle multiple documents. Transformer models already struggle to handle a full single document at a time. Requiring them to handle multiple potentially long documents at once remains an open problem. We present two current approaches in the following sections to modifying Transformers for multiple-documents, but stress that this remains an open problem. These approaches are a two-step summarization process and hierarchical Transformers.

5.2 Two-Step Summarization

Both the WikiSum and Multi-News publications present a two-step summarization process applied to their respective datasets which reduces the multiple-document summarization task to single-document summarization. First, each individual document in the cluster is reduced down to their most important sentences. For Multi-news, this involves truncating the article to the first few sentences, as the expectation is that the first few sentences are the most important. In WikiSum, they use extractive summarization methods like LexRank to determine important sentences in the individual documents. Once the important sentences in the documents are identified, they are concatenated together into a single new super-document. Other sentences are effectively ignored.

The second step applies a single-document summarization model to the super-document. Thus, the multiple-document summarization problem is reduced to a single-document summarization model. The most successful two-step systems use a Transformer model, however any single-document system can be used. For the Transformer model, it is important that aggressive filtering of important sentences is used in the first step, to ensure that the super-document remains short enough to fit within the attention mechanism.

5.3 Hierarchical Transformers

The first step in the two-step summarization process is a ranking step. Ranking multiple-document data to determine most important data makes sense: multiple-document data is hierarchical by nature. Words make up sentences, sentences make up paragraphs, paragraphs make up documents, and the documents make up the full example. This ranked nature should be taken advantage of; the summarization model ideally is aware of the nature of the data. Thus, another solution to multiple-document summarization is to incorporate hierarchical structure directly into the summarization model.

We detail two different methods for incorporating the hierarchical nature of the problem into a Transformer model. Liu and Lapata (2019) [23] propose a Transformer with three different attention layers: local, global, and graph attention. Local attention is exactly the attention mechanism already found in the traditional Transformer architecture, and encodes each input text span individually. Global attention is applied across text spans, allowing for the exchange of information across the input paragraphs (see figure 7). Graph attention is built from a separate input entirely, a similarity graph (as in LexRank) or discourse relation graph [8] representing the document collection. The graph attention, designed to capture inter-document relationships, then replaces the weights of one of the global attention layers, essentially combining global and graph attention.

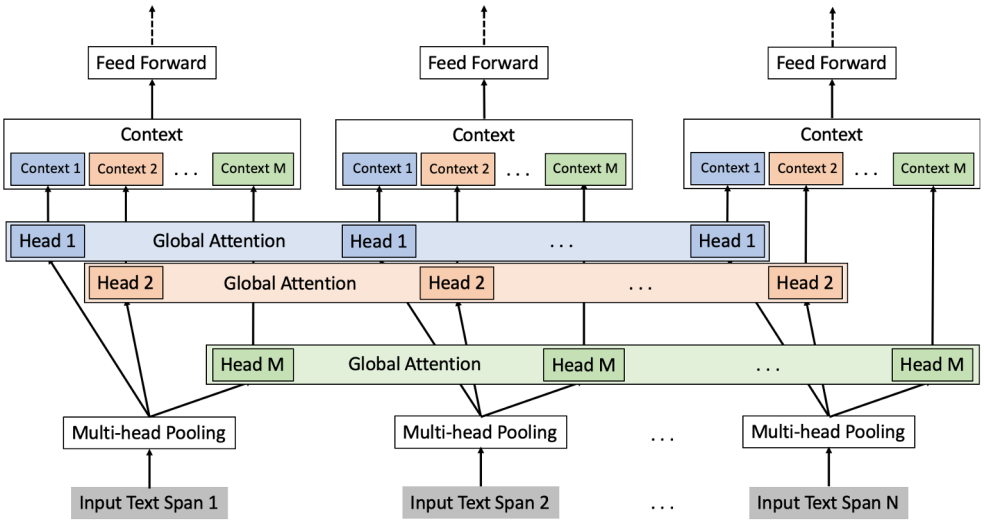

Figure 7: Global Attention as Defined in Liu and Lapata (2019)

Liu and Lapata find that this three-attention hierarchical Transformer significantly outperforms 'flat' Transformers that try to model everything at once, as the Transformer is applied in two-step summarization approaches. However, they find that the graph attention is mostly redundant, as the inter-paragraph global attention already captures most inter-document information. Li et al. (2020) [17] show that a more direct incorporation of graph attention into the encoding process significantly improves performance and removes this redundancy (see figure 8). Similar to Liu and Lapata's split in local and global attention, Li et al. use a local and global split to graph attention. The local graph attention considers only pairwise relationships of the current node and is added to local attention, allowing the input paragraph encodings to consider cross-paragraph and cross-document relationships of nearby nodes. Global graph attention, which considers the entirety of the graph, is applied to summary decoding to guide the generation process.

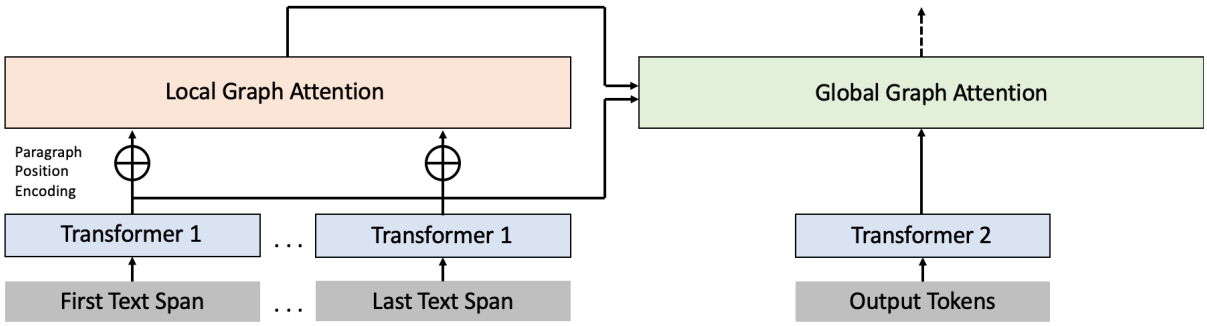

Figure 8: Adding Graph Attention to Transformers as in Li et. al (2020)

Both solutions essentially reflect the hierarchical nature of the data into the Transformer attention mechanism. Local attention remains focused on producing encodings for input text spans. Global attention is added to explicitly model the relationships between these inputs, modeling the hierarchical nature of the data. These solutions are particularly attractive as they do not require the user to explicitly rank the input data themselves and ignore all but the highest ranked. All data is used as the input for the hierarchical model, and it implicitly learns to rank the data and attend only to the most important text spans.

6 System Recommendations

6.1 Single-Document Summarization

Our recommendations for single-document summarization are directly tied to available resources. A consistent theme with summarization models is that the better the model becomes, the more computational and data resources it requires to build and run. The earliest extractive solutions effectively require nothing in advance. They can be built without any labeled data, and some variations require no unlabeled data either; the variations that do require labeled data can often be trained on the target documents to summarize, or generic monolingual corpora. These systems also require no specialized hardware, running relatively quickly on CPU.

As labeled data becomes available, either because the user already has labeled data or their domain matches one of the publicly available datasets, more sophisticated neural models become an option. At this point, which neural model to choose is almost exclusively based on compute requirements, and a choice between extractive and abstractive summarization. Transformers, while offering the best performance, require heavy GPU use to train and also to evaluate in reasonable time. They also require the most examples to train adequately. If the requirements are met, we highly recommend using a Transformer whenever possible.

If labeled data exists, but either the computational costs of Transformers are too high or the Transformer fails for another reason, attentional-RNNs are a good fallback. Both extractive and abstractive attentional-RNNs exist, and both are valid options. While the abstractive RNNs generally outperform the extractive systems in raw overall score, abstractive systems have a tendency to catastrophically fail on some individual inputs. This failure can look like infinite repetition of a single unigram or trigram in output, or output that just reads as unintelligible nonsense. If the user prioritizes a minimum performance on each summary, the extractive systems would be a better choice, as they always output a summary that is at least as grammatical and fluent as the input. In cases where overall performance over all documents is the priority, abstractive RNNs will do better.

6.2 Multiple-Document Summarization

Early efforts into multiple-document summarization by the MILU project has focused on wrapping our single-document Transformer models into the two-step summarization process. Unlike the models reported in WikiSum [22] or Multi-News [12], we have not found success porting single-document models into this

pipeline. Our early results appear to indicate that standard Transformers are insufficient and modifications to the model are needed.

Analyzing the models used in WikiSum and Multi-News, we find that neither used standard Transformers; both included novel modifications that may help the Transformer be more adaptable to the two-step process. WikiSum modified their attention mechanism in order to accommodate longer inputs, whereas Multi-News replaces a random attention head in their Transformer with a copy mechanism. It is possible that these changes allowed these single-document Transformers to better transfer to the new problem space.

Arguably, if specialized model features are required for models to be usable in this solution structure, it is preferable to use the hierarchical models which incorporate the ranking nature of the data into the model itself. If users have a successful single-document summarization model, it is possible that they can use the two-step process with their current model; however, success is not guaranteed. If this two-step process fails, or the user does not already have a high-quality single-document summarization Transformer, we recommend trying one of the hierarchical Transformer solutions instead. However, these solutions for multi-document summarization are largely based on newly published research; it is possible that they will not be available for general use in the near future.

Both the two-step process and the hierarchical models require significant compute power and labeled data; in many situations just one or possibly neither of these requirements will be satisfied. In the case where the user does not have any labeled data, or little compute power, using graph algorithms may be the only option, despite their inherent weaknesses in multiple-document summarization. With minimal labeled data, but a budget for high amounts of compute power, it is possible a Transformer pretrained with a self-supervised approach such as PEGASUS [50] can be used in either the two-step process or adapted into a hierarchical model. However, we are unaware of such a model existing or being publicly available, and thus this would require active development.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014.

[2] Satanjeev Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[3] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020.

[4] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. *CoRR*, abs/1508.05326, 2015.

[5] Jianpeng Cheng and Mirella Lapata. Neural summarization by extracting sentences and words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 484–494, Berlin, Germany, August 2016. Association for Computational Linguistics.

[6] Davide Chicco. *Siamese Neural Networks: An Overview*, pages 73–94. Springer US, New York, NY, 2021.

[7] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers, 2020.

[8] Janara Christensen, Mausam, Stephen Soderland, and Oren Etzioni. Towards coherent multi-document summarization. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1163–1173, Atlanta, Georgia, June 2013. Association for Computational Linguistics.

[9] Anirban Dasgupta, Ravi Kumar, and Sujith Ravi. Summarization through submodularity and dispersion. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1014–1022, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[11] Günes Erkan and Dragomir R. Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *CoRR*, abs/1109.2128, 2011.

[12] Alexander Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir Radev. 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*, pages 1074–1084, Florence, Italy, July 2019. Association for Computational Linguistics.

[13] Ari Holtzman, Jan Buys, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *CoRR*, abs/1904.09751, 2019.

[14] Kun Jing and Jungang Xu. A survey on neural network language models, 2019.

[15] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. In *Journal of Documentation, Vol. 28 No. 1*, pages 11–21, 1972.

[16] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer, 2020.

[17] Wei Li, Xinyan Xiao, Jiachen Liu, Hua Wu, Haifeng Wang, and Junping Du. Leveraging graph to improve abstractive multi-document summarization, 2020.

[18] Yanyang Li, Tong Xiao, Yinqiao Li, Qiang Wang, Changming Xu, and Jingbo Zhu. A simple and effective approach to coverage-aware neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 292–297, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[19] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[20] Hui Lin and Jeff Bilmes. Multi-document summarization via budgeted maximization of submodular functions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 912–920, Los Angeles, California, June 2010. Association for Computational Linguistics.

[21] Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 510–520, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[22] Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. *CoRR*, abs/1801.10198, 2018.

[23] Yang Liu and Mirella Lapata. Hierarchical transformers for multi-document summarization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5070–5081, Florence, Italy, July 2019. Association for Computational Linguistics.

[24] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[25] Inderjeet Mani. *Automatic Summarization*, volume 3. 06 2001.

[26] Qiaozhu Mei, Jian Guo, and Dragomir Radev. Divrank: the interplay of prestige and diversity in information networks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1009–1018, 07 2010.

[27] Rada Mihalcea and Paul Tarau. TextRank: Bringing order into text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 404–411, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[28] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[29] Ramesh Nallapati, Bing Xiang, and Bowen Zhou. Sequence-to-sequence rnns for text summarization. *CoRR*, abs/1602.06023, 2016.

[30] Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. *CoRR*, abs/1611.04230, 2016.

[31] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar GuÌ‡lçehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany, August 2016. Association for Computational Linguistics.

[32] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *CoRR*, abs/1808.08745, 2018.

[33] Ani Nenkova and Rebecca Passonneau. Evaluating content selection in summarization: The pyramid method. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pages 145–152, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics.

[34] Paul Over and James Yen. An introduction to duc2004. In *Proceedings of the 4th Document Understanding Conference (DUC 2004)*, 2004.

[35] Larry Page, Sergey Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web, 1998.

[36] Matteo Pagliardini, Prakhar Gupta, and Martin Jaggi. Unsupervised learning of sentence embeddings using compositional n-gram features. *CoRR*, abs/1703.02507, 2017.

[37] Maxime Peyrard. Studying summarization evaluation metrics in the appropriate scoring range. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5093–5100, Florence, Italy, July 2019. Association for Computational Linguistics.

[38] Dragomir R. Radev, Hongyan Jing, and Malgorzata Budzikowska. Centroid-based summarization of multiple documents: sentence extraction utility-based evaluation, and user studies. *CoRR*, cs.CL/0005020, 2000.

[39] A. Radford, Jeffrey Wu, R. Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019.

[40] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019.

[41] Alexander M. Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[42] Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. *CoRR*, abs/1704.04368, 2017.

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[44] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks, 2015.

[45] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[46] Michihiro Yasunaga, Rui Zhang, Kshitijh Meelu, Ayush Pareek, Krishnan Srinivasan, and Dragomir Radev. Graph-based neural multi-document summarization. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 452–462, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[47] Yongjing Yin, Linfeng Song, Jinsong Su, Jiali Zeng, Chulun Zhou, and Jiebo Luo. Graph-based neural sentence ordering. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 5387–5393. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[48] Dani Yogatama, Fei Liu, and Noah A. Smith. Extractive summarization by maximizing semantic volume. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1961–1966, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[49] Wenyuan Zeng, Wenjie Luo, Sanja Fidler, and Raquel Urtasun. Efficient summarization with read-again and copy mechanism. *CoRR*, abs/1611.03382, 2016.

[50] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization, 2019.

[51] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with BERT. *CoRR*, abs/1904.09675, 2019.

[52] Xiaojin Zhu, Andrew Goldberg, Jurgen Van Gael, and David Andrzejewski. Improving diversity in ranking using absorbing random walks. *HLT-NAACL*, pages 97–, 01 2007.