KENNESAW STATE UNIVERSITY

Final Project:

Text Summarization

CS 4742

Natural Language Processing

Fall 2024

Instructor – Dr. Michail Alexiou

Michael Rizig, Sam Bostian, Colton Baldwin

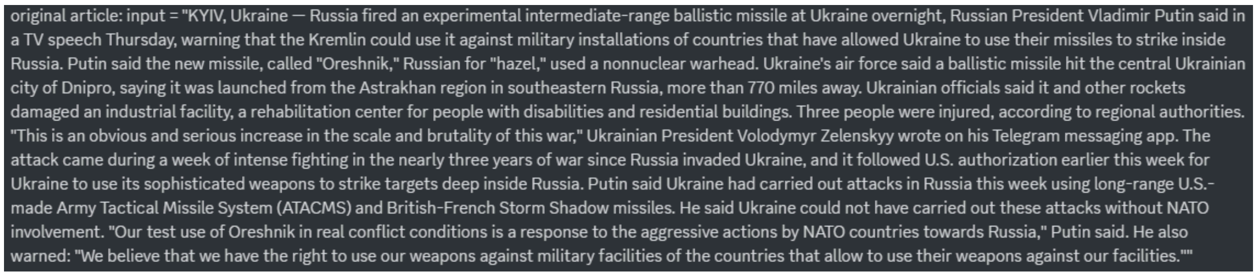
Table of Contents

I. Methodology

Abstract

Text Summarization is a natural language processing technique that analyzes copious amounts of text and condenses it into a concise summarization containing vital information from the original text. This NLP technique is important because it saves users time when they need to obtain a quick high-level summation of important information. This project is a study of multiple modern NLP models and how well they are able to handle the task of text summarization. Four separate models were used to generate summaries from the CNN/Daily Mail data set. The model used to create these summaries includes a custom-created extractive summarizer that uses a bag of words technique to select the best sentences in the article. A fine-tuned BART model that was fine-tuned using the CNN/Daily Mail data set to create an abstractive summary. A combination of a BERT model encodes the text and then passes the vectors created into the fine-tuned BART model to produce an abstractive summary. A T5-small model was used for a comparison base to the other models' abstractive summaries.

Extractive versus Abstractive



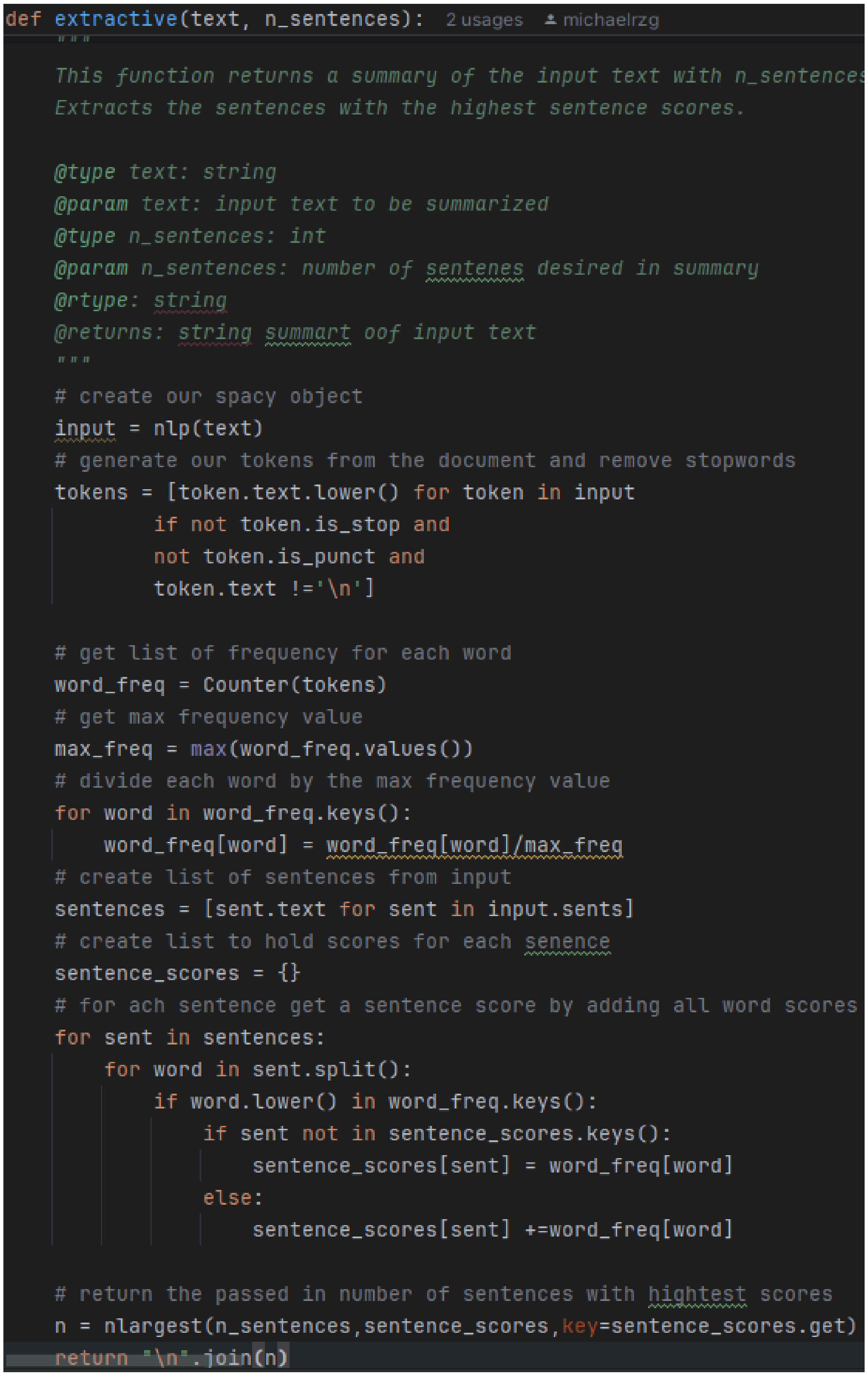
```
original article: input = "KYIV, Ukraine — Russia fired an experimental intermediate-range ballistic missile at Ukraine overnight, Russian President Vladimir Putin said in a TV speech Thursday, warning that the Kremlin could use it against military installations of countries that have allowed Ukraine to use their missiles to strike inside Russia. Putin said the new missile, called "Oreshnik," Russian for "hazel," used a nonnuclear warhead. Ukraine's air force said a ballistic missile hit the central Ukrainian city of Dnipro, saying it was launched from the Astrakhan region in southeastern Russia, more than 770 miles away. Ukrainian officials said it and other rockets damaged an industrial facility, a rehabilitation center for people with disabilities and residential buildings. Three people were injured, according to regional authorities. "This is an obvious and serious increase in the scale and brutality of this war," Ukrainian President Volodymyr Zelenskyy wrote on his Telegram messaging app. The attack came during a week of intense fighting in the nearly three years of war since Russia invaded Ukraine, and it followed U.S. authorization earlier this week for Ukraine to use its sophisticated weapons to strike targets deep inside Russia. Putin said Ukraine had carried out attacks in Russia this week using long-range U.S.-made Army Tactical Missile System (ATACMS) and British-French Storm Shadow missiles. He said Ukraine could not have carried out these attacks without NATO involvement. "Our test use of Oreshnik in real conflict conditions is a response to the aggressive actions by NATO countries towards Russia," Putin said. He also warned: "We believe that we have the right to use our weapons against military facilities of the countries that allow to use their weapons against our facilities.""
```

Extractive summarization involves picking the most relevant sentences to the topic of an article. Then these sentences are reorganized to form a comprehensive summary. These sentences are taken verbatim from the original article. There are three fundamental operations to extractive summarization: processing the text into a numerical format, scoring the processed sentences, and selecting the k number of sentences with the most significant scores. The extractive model processes the text to remove insignificant words. Once each meaningful token is isolated a score is based on the frequency found in the text. Then each sentence receives a score based on the words found in the sentence. The words with the top scores are selected and they are placed in the order they are found in the text to create a smoother summary.

```
Extractive Summary:

KYIV, Ukraine — Russia fired an experimental intermediate-range ballistic missile at Ukraine
overnight, Russian President Vladimir Putin said in a TV speech Thursday, warning that the Kremlin
could use it against military installations of countries that have allowed Ukraine to use their
missiles to strike inside Russia.
The attack came during a week of intense fighting in the nearly three years of war since Russia
invaded Ukraine, and it followed U.S. authorization earlier this week for Ukraine to use its
sophisticated weapons to strike targets deep inside Russia.
He also warned: "We believe that we have the right to use our weapons against military facilities of
the countries that allow to use their weapons against our facilities."
Ukraine's air force said a ballistic missile hit the central Ukrainian city of Dnipro, saying it was
launched from the Astrakhan region in southeastern Russia, more than 770 miles away.
```

Abstractive summarization creates unique sentences that summarize an article from vital information found within the original text. These summaries should be arranged in a logical, and grammatically correct manner. There are two approaches to abstractive summarization: structured-based and semantic-based. A structured-based summary capitalizes on the original text's structure to identify, and then extract meaningful information based on frequency. Finally, it organizes this information into a summarization. For a semantic-based, the relationship and context of tokens found in the document are analyzed to produce a unique summary that accurately captures the connotation of the original text. The main difference between structure-based and semantic-based approaches is that structure-based focuses on the frequency of words similar to extractive summarizations. Whereas, semantic-based summarization focuses on the context of the tokens found in the text to provide an accurate summarization of the original text.

```
Abstractive Summary with BART:

Russia fires intermediate-range ballistic missile at Ukraine, President Vladimir Putin says. Putin
says the new missile, called "Oreshnik," used a nonnuclear warhead. Ukraine's air force said a
ballistic missile hit the central Ukrainian city of Dnipro.



Abstractive Summary with BERT + BART:
1ize : kyiv, ukraine — russia fired an experimental intermediate - range ballistic missile at
ukraine overnight, russian president vladimir putin said in a tv speech thursday, warning that the
kremlin could use it against military installations of countries that have allowed ukraine to use
their missiles to strike inside russia. putin said the missile, called " oreshnik, " russian for "
hazel, " used a nonnuclear warhead. ukraine's air force said a ballistic missile hit the central
ukrainian city of dnipro, saying it was launched from the astrakhan region in southeastern russia,
3 1



Abstractive Summary with  T5:
the missile, called "Oreshnik," used a nonnuclear warhead. the missile hit the central Ukrainian
city of Dnipro. the attack came during a week of intense fighting in the nearly three years of war.
```

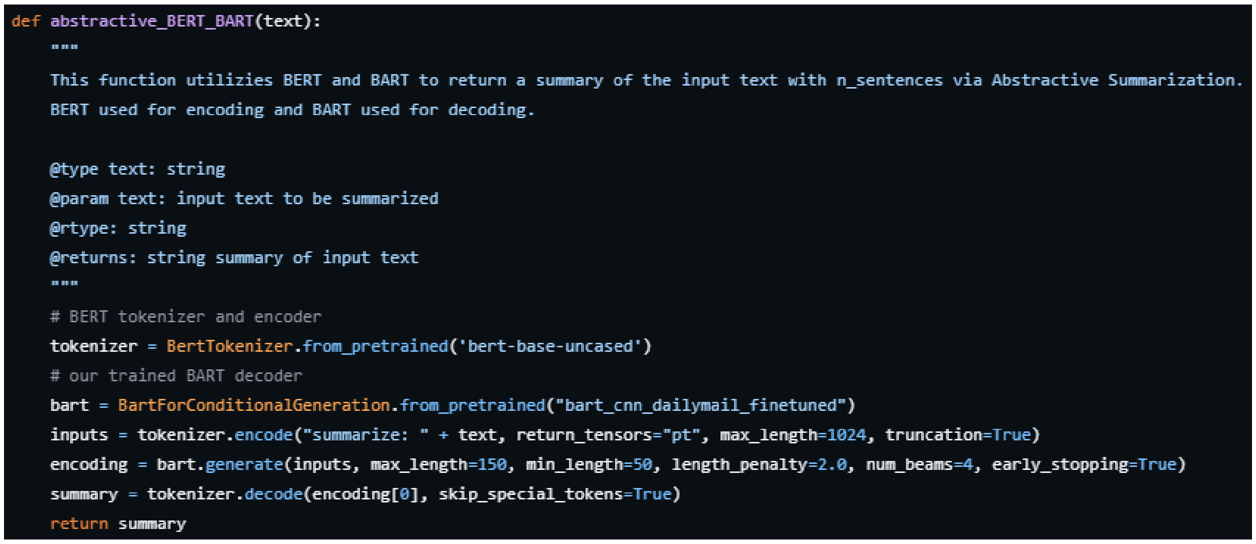


```
def extractive(text, n_sentences):  2 usages  michaelrzg
    """
    This function returns a summary of the input text with n_sentences
    Extracts the sentences with the highest sentence scores.

    @type text: string
    @param text: input text to be summarized
    @type n_sentences: int
    @param n_sentences: number of sentenes desired in summary
    @rtype: string
    @returns: string summart oof input text
    """
    # create our spacy object
    input = nlp(text)
    # generate our tokens from the document and remove stopwords
    tokens = [token.text.lower() for token in input
            if not token.is_stop and
            not token.is_punct and
            token.text !='\n']

    # get list of frequency for each word
    word_freq = Counter(tokens)
    # get max frequency value
    max_freq = max(word_freq.values())
    # divide each word by the max frequency value
    for word in word_freq.keys():
        word_freq[word] = word_freq[word]/max_freq
    # create list of sentences from input
    sentences = [sent.text for sent in input.sents]
    # create list to hold scores for each senence
    sentence_scores = {}
    # for ach sentence get a sentence score by adding all word scores
    for sent in sentences:
        for word in sent.split():
            if word.lower() in word_freq.keys():
                if sent not in sentence_scores.keys():
                    sentence_scores[sent] = word_freq[word]
                else:
                    sentence_scores[sent] +=word_freq[word]

    # return the passed in number of sentences with hightest scores
    n = nlargest(n_sentences,sentence_scores,key=sentence_scores.get)
    return "\n".join(n)
```

Figure 1a. Extractive Summarization Implementation

BERT Model

First introduced to the computer science community in 2018 from the paper, BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding. BERT is Google AI's Jacob Delvin and his team's answer to the problem that previous transformer models had with losing the context of tokens during embeddings. BERT stands for Bidirectional Encoder Representation Transformer and revolutionized deep learning models. Unlike other encoder or decoder models that can only analyze self-attention from the left or right of a token. BERT is capable of analyzing the context of a token from both directions of a sentence to understand the context of a token. Tokens are fed into the encoder, where they are converted into context vectors to be processed by the neural network. A sequence of vectors that correspond to the input token is produced to be evaluated for context. Masking is performed by the model so that words do not lose context during training. BERT-base contains 110 million parameters and BERT-base has 340 million parameters.

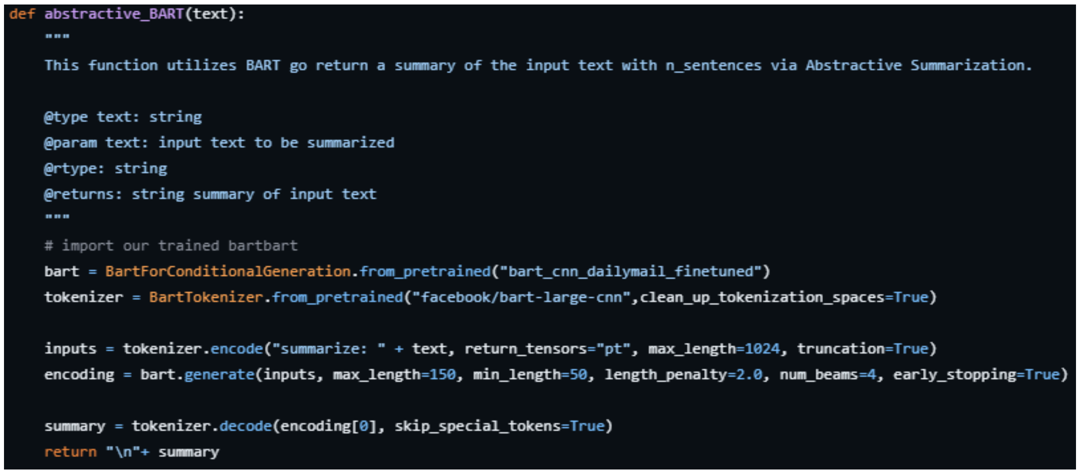


```python
def abstractive_BERT_BART(text):
    """
    This function utilizies BERT and BART to return a summary of the input text with n_sentences via Abstractive Summarization.
    BERT used for encoding and BART used for decoding.

    @type text: string
    @param text: input text to be summarized
    @rtype: string
    @returns: string summary of input text
    """
    # BERT tokenizer and encoder
    tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
    # our trained BART decoder
    bart = BartForConditionalGeneration.from_pretrained("bart_cnn_dailymail_finetuned")
    inputs = tokenizer.encode("summarize: " + text, return_tensors="pt", max_length=1024, truncation=True)
    encoding = bart.generate(inputs, max_length=150, min_length=50, length_penalty=2.0, num_beams=4, early_stopping=True)
    summary = tokenizer.decode(encoding[0], skip_special_tokens=True)
    return summary
```

Figure 1b. Abstractive BERT plus BART Summarization Implementation

BART Model

Not to be outdone by Google AI; Facebook AI introduced the BART model in a paper submitted in 2019 titled, BART: Denoising Sequence-To-Sequence Pre-Training for Natural Language Generation, Translation and Comprehension. The BART model was created as a response to the BERT model and to improve on Google AI's ground breaking work. Two major components were added to the encoder model principles that BERT was founded on. The first component is that BART was trained as a denoising autoencoder along with the traditional masking that BERT uses. The model is trained by corrupting an input document and then rebuilding the document. To evaluate and optimize the rebuilding of the original document a reconstruction loss is calculated based on the cross-entropy between the decoder's output and the original document. The corruption methods used besides basic masking techniques included token deletion, text infilling, sentence permutation and document rotation. Document rotation is achieved by selecting a new starting token to begin. The other major improvement used was the addition of an autoregressive that predicts the next predicted token based on the previous input. So BART takes the bidirectional encoder from the BERT model and uses Open AI's GPT decoder to generate the summarization. BART has 140 million parameters and was trained on 160 GBs of data. To achieve a specific NLP task the BART model must be fine-tuned to perform tasks like translation or summarization.

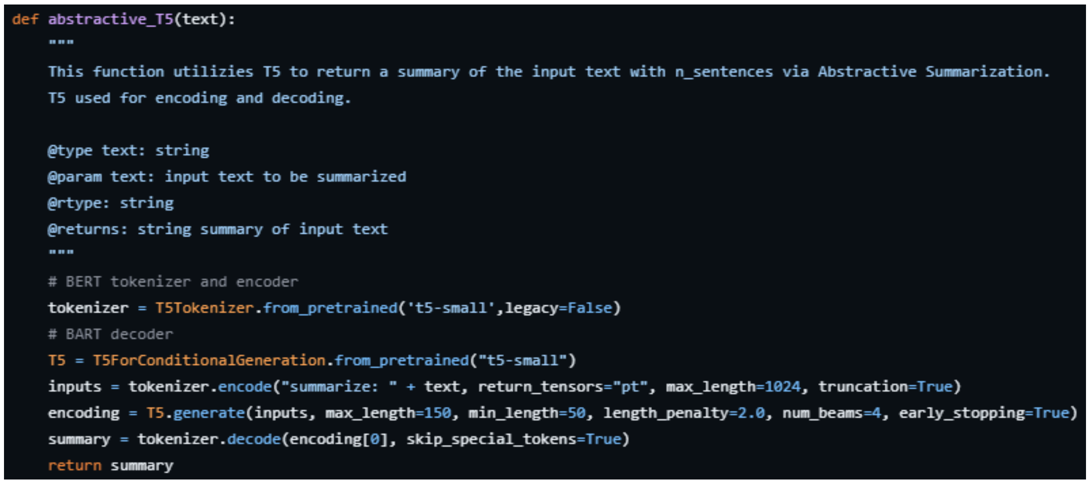


```python
def abstractive_BART(text):
    """
    This function utilizes BART go return a summary of the input text with n_sentences via Abstractive Summarization.

    @type text: string
    @param text: input text to be summarized
    @rtype: string
    @returns: string summary of input text
    """
    # import our trained bartbart
    bart = BartForConditionalGeneration.from_pretrained("bart_cnn_dailymail_finetuned")
    tokenizer = BartTokenizer.from_pretrained("facebook/bart-large-cnn",clean_up_tokenization_spaces=True)

    inputs = tokenizer.encode("summarize: " + text, return_tensors="pt", max_length=1024, truncation=True)
    encoding = bart.generate(inputs, max_length=150, min_length=50, length_penalty=2.0, num_beams=4, early_stopping=True)

    summary = tokenizer.decode(encoding[0], skip_special_tokens=True)
    return "\n"+ summary
```

Figure 1c. Abstractive BART Summarization Implementation

Text-to-Text Transfer Transformer (T5)

T5 is a unified NLP model architecture because it uses the text-to-text paradigm. This paradigm is based on a learning approach where text is imputed into a model and a corresponding output text is generated. This is the same as the BART model and also is formed off of the encoder-decoder model but instead of writing an entire new sentence only fills in the masked components. The T5 model was trained on the Colossal Clean Crawled Corpus which is 750 GB. T5 can come in many different sizes: T5-small with about 60 million parameters, t5-base 220 with about million parameters, T5-large with about 770 million parameters, T5-3B with 3 billion parameters, and T%-11B with 11 billion parameters.

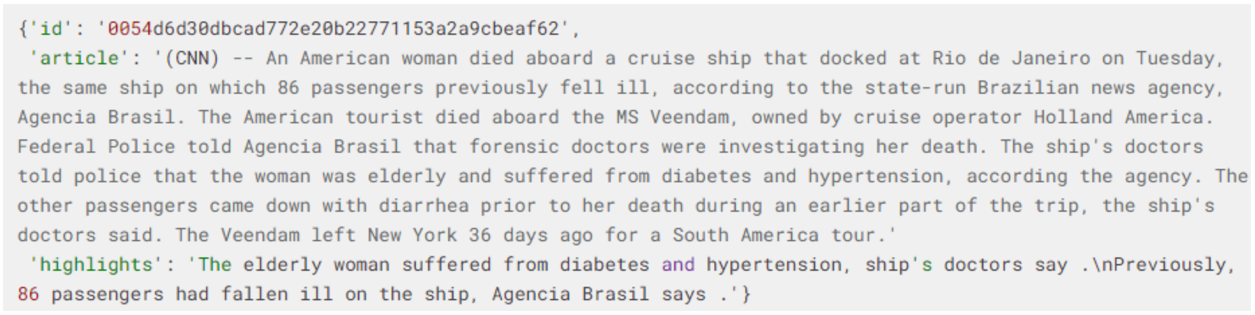


```python
def abstractive_T5(text):
    """
    This function utilizes T5 to return a summary of the input text with n_sentences via Abstractive Summarization.
    T5 used for encoding and decoding.

    @type text: string
    @param text: input text to be summarized
    @rtype: string
    @returns: string summary of input text
    """
    # BERT tokenizer and encoder
    tokenizer = T5Tokenizer.from_pretrained('t5-small',legacy=False)
    # BART decoder
    T5 = T5ForConditionalGeneration.from_pretrained("t5-small")
    inputs = tokenizer.encode("summarize: " + text, return_tensors="pt", max_length=1024, truncation=True)
    encoding = T5.generate(inputs, max_length=150, min_length=50, length_penalty=2.0, num_beams=4, early_stopping=True)
    summary = tokenizer.decode(encoding[0], skip_special_tokens=True)
    return summary
```

Figure 1d. Abstractive T5 Summarization Implementation.

CNN/Daily Mail Dataset

The CNN/Daily Mail data set is a common data set used for training NLP models. The original version, 1.0.0, was developed for machine reading comprehension and question-and-answer problems. The two later versions of the data set were adjusted for text summarization. The corpus contains over three hundred thousand articles written between June 2010 and April 2015. The data set consists of about two hundred and eighty-seven thousand training sets with thirteen thousand validation sets and eleven thousand testing sets. There are three data fields in this set: an id, the article, and the highlights. The id is a string that is the web address for the original article stored as a hexadecimal formatted as SHA1 hash. The article is one string containing the news article. The highlights is an author-created summarization containing the highlights of the article. There is a mean of 781 tokens in articles with that number being 56 for highlights.



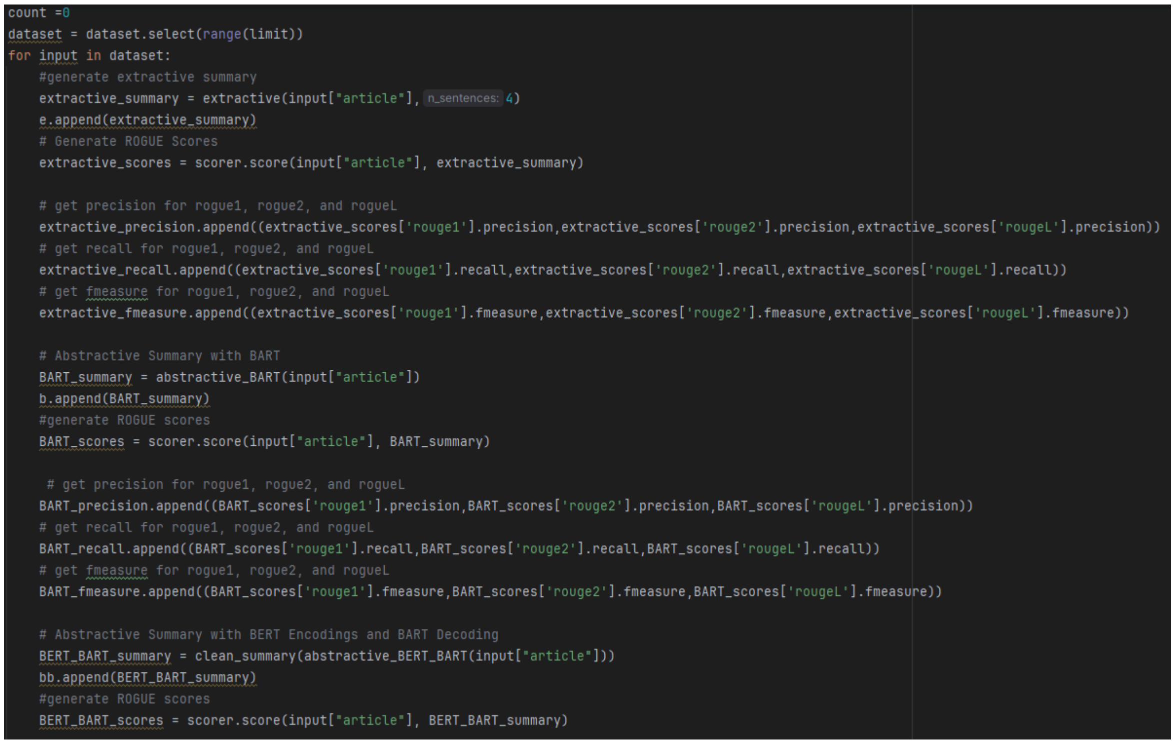
```
{'id': '0054d6d30dbcad772e20b22771153a2a9cbeaf62',
 'article': '(CNN) -- An American woman died aboard a cruise ship that docked at Rio de Janeiro on Tuesday,
the same ship on which 86 passengers previously fell ill, according to the state-run Brazilian news agency,
Agencia Brasil. The American tourist died aboard the MS Veendam, owned by cruise operator Holland America.
Federal Police told Agencia Brasil that forensic doctors were investigating her death. The ship's doctors
told police that the woman was elderly and suffered from diabetes and hypertension, according the agency. The
other passengers came down with diarrhea prior to her death during an earlier part of the trip, the ship's
doctors said. The Veendam left New York 36 days ago for a South America tour.'
 'highlights': 'The elderly woman suffered from diabetes and hypertension, ship's doctors say .\nPreviously,
86 passengers had fallen ill on the ship, Agencia Brasil says .'}
```

Figure 1e. CNN/DailyMail Dataset Sample

Evaluation Techniques: ROUGE and BART Score

Recall-Oriented Understudy for Gisting Evaluation, commonly known as ROUGE, is a set of metrics designed to evaluate NLP text summarization. ROUGE-N can be used to determine the number of overlaps of n-grams between the generated summary and a sampled text. Another version of ROUGE is ROUGE-L, this version uses the longest common subsequence. This subsequence does not necessarily be consecutive in the generated text but must be in order when compared to the original text's token placement. The ROUGE is used to calculate the precision, recall, and F1 score for the summarizing model. The precision is used to determine how relevant the summarization is to the original text. A high precision score would indicate that there is a high repeat of the original text in the summary. Recall is used to measure the amount of the original text that has been duplicated in the summary. The F1 score uses both recall and precision to create a balanced metric that accounts for the false positives and negatives that a model creates.

Another method for evaluating the performance of NLP text summarization is the BERT score. BERT score evaluation is better at assessing the context of a generated summary compared to n-gram evaluation which only evaluates token placement. The BERT score uses a BERT model to convert each text into a context vector of both the original text and the generated summary. Those two vectors are then compared by cosine similarity to see how well the model is capable of producing concise summaries. To calculate the similarity the dot product of the two vectors is divided by the product of the magnitude of both vectors. The closer this calculation is to one the greater the similarity of the two vectors.



```
count =0
dataset = dataset.select(range(limit))
for input in dataset:
    #generate extractive summary
    extractive_summary = extractive(input["article"], n_sentences: 4)
    e.append(extractive_summary)
    # Generate ROGUE Scores
    extractive_scores = scorer.score(input["article"], extractive_summary)

    # get precision for rogue1, rogue2, and rogueL
    extractive_precision.append((extractive_scores['rouge1'].precision,extractive_scores['rouge2'].precision,extractive_scores['rougeL'].precision))
    # get recall for rogue1, rogue2, and rogueL
    extractive_recall.append((extractive_scores['rouge1'].recall,extractive_scores['rouge2'].recall,extractive_scores['rougeL'].recall))
    # get fmeasure for rogue1, rogue2, and rogueL
    extractive_fmeasure.append((extractive_scores['rouge1'].fmeasure,extractive_scores['rouge2'].fmeasure,extractive_scores['rougeL'].fmeasure))

    # Abstractive Summary with BART
    BART_summary = abstractive_BART(input["article"])
    b.append(BART_summary)
    #generate ROGUE scores
    BART_scores = scorer.score(input["article"], BART_summary)

     # get precision for rogue1, rogue2, and rogueL
    BART_precision.append((BART_scores['rouge1'].precision,BART_scores['rouge2'].precision,BART_scores['rougeL'].precision))
    # get recall for rogue1, rogue2, and rogueL
    BART_recall.append((BART_scores['rouge1'].recall,BART_scores['rouge2'].recall,BART_scores['rougeL'].recall))
    # get fmeasure for rogue1, rogue2, and rogueL
    BART_fmeasure.append((BART_scores['rouge1'].fmeasure,BART_scores['rouge2'].fmeasure,BART_scores['rougeL'].fmeasure))

    # Abstractive Summary with BERT Encodings and BART Decoding
    BERT_BART_summary = clean_summary(abstractive_BERT_BART(input["article"]))
    bb.append(BERT_BART_summary)
    #generate ROGUE scores
    BERT_BART_scores = scorer.score(input["article"], BERT_BART_summary)
```

Figure 1f. Metric Generation Implementation

II. Results

To test our models, we utilized 3 scoring criteria: Precision, Recall, and F1 score. We then measured these criteria across our 4 different summarization metrics, ROGUE-1, ROGUE-2, ROGUE-L, and BERTScore. The table in Figure 2a shows our testing results for each of these metrics.

Models	**ROGUE-1**			**ROGUE-2**			**ROGUE-L**			**BERTScore**		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Extractive	1.0	.309	.445	.976	.299	.432	.658	.191	.281	.823	.635	.716
BART	.984	.121	.206	.837	.102	.174	.870	.107	.183	.834	.511	.633
BERT + BART	.932	.255	.362	.916	.257	.352	.931	.254	.360	.841	.609	.703
T5	.956	.110	.189	.785	.090	.155	.859	.102	.174	.815	.476	.600

Figure 2a. Results of All Metrics

Figure 2b graphically represents our BERTScore data for the 4 models. We see that all models perform relatively similarly but Extractive has the highest recall, and our BERT BART combination outperforms the standalone BART model. T5 Scores the worst across all metrics.

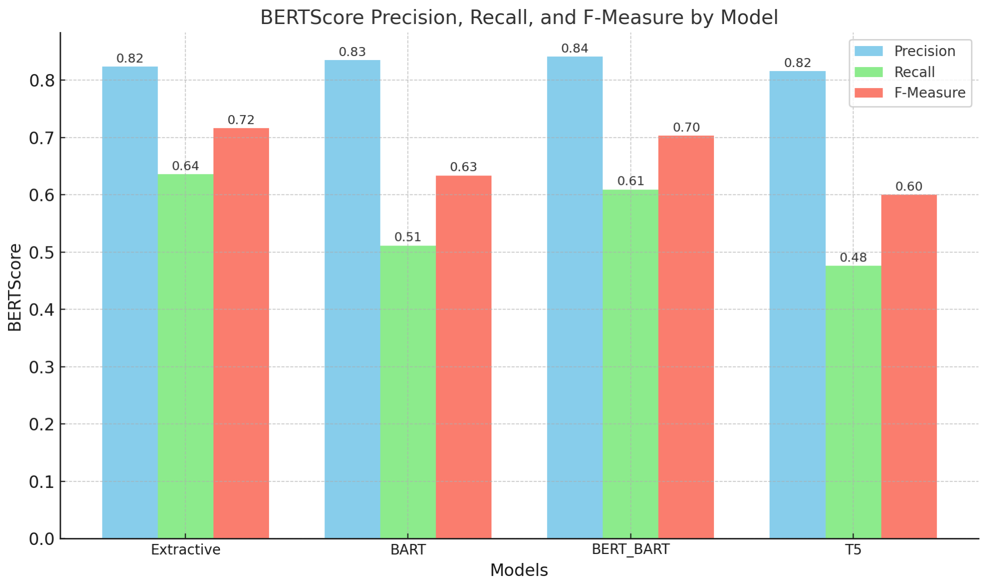


Figure 2b. BERTScore Comparison of Precision, Recall, and F1 Score Across All 4 Models

Figure 2C represents our F1 Scores across our 4 models in all 3 metrics. We see that Extractive scores highly in ROGUE-1 and ROGUE-2 but is outperformed in ROGUE-L. We also see that once again BERT + BART combination scores significantly higher than just BART across all 3 metrics. T5 once again scores the worst.

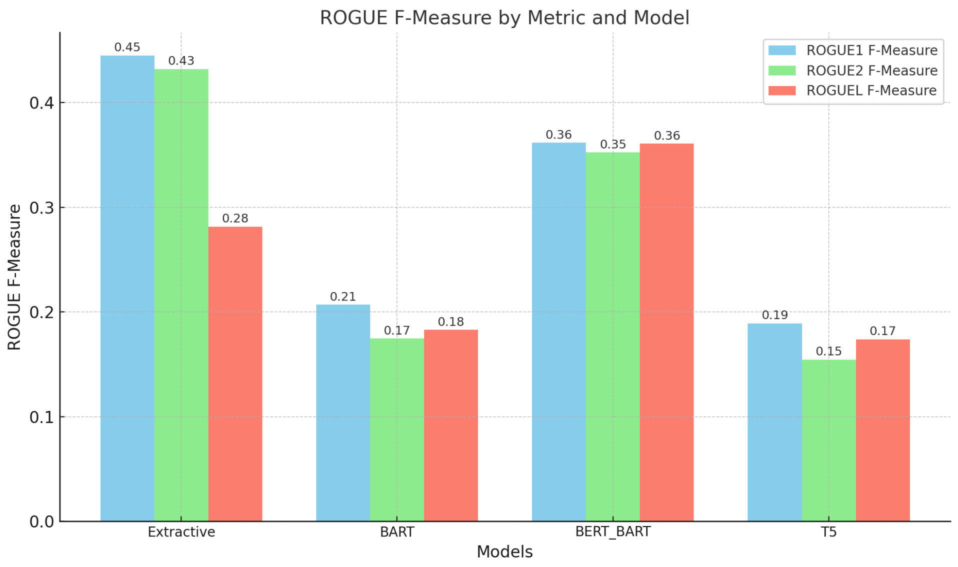


Figure 2c. F1 Score Comparison of ROGUE 1, 2, and L Metrics Across All 4 Models

Figure 2d represents the recall scores for all 4 models across our 3 metrics. We see once again that our BERT BART combination outperforms the singular BART by a margin of nearly two times. T5 scores similarly to BART and Extractive scores the highest.

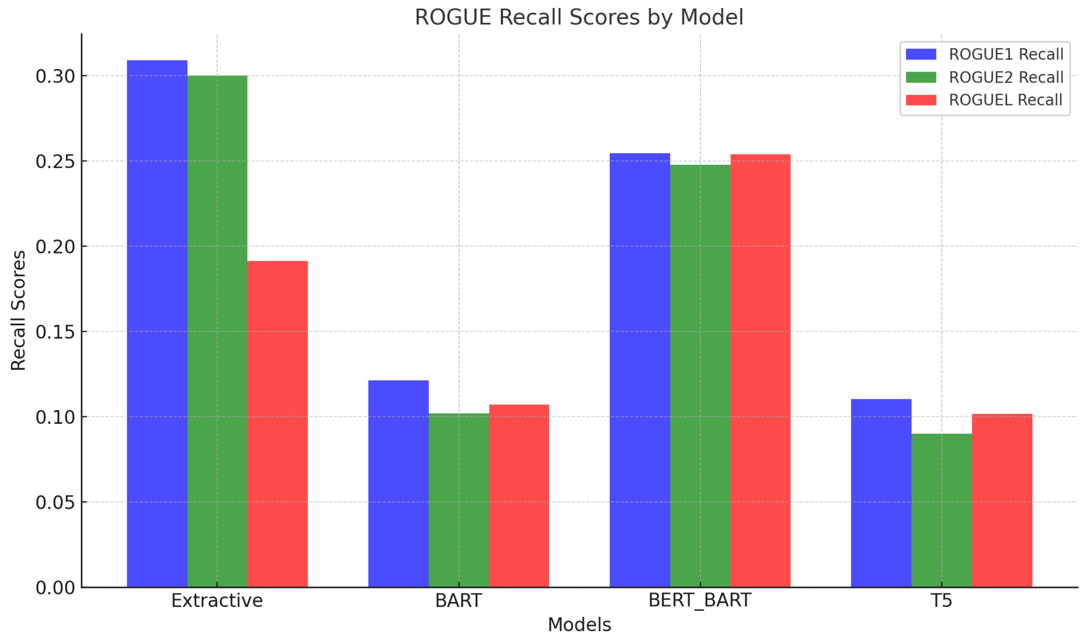


Figure 2d. Recall Comparison of ROGUE 1, 2, and L Metrics Across All 4 Models

Finally, Figure 2e shows precision scores for all models across the ROGUE metrics. We can see that the models score relatively the same across this metric, with a noticeable dip in precision from our extractive ROGUE-L score.

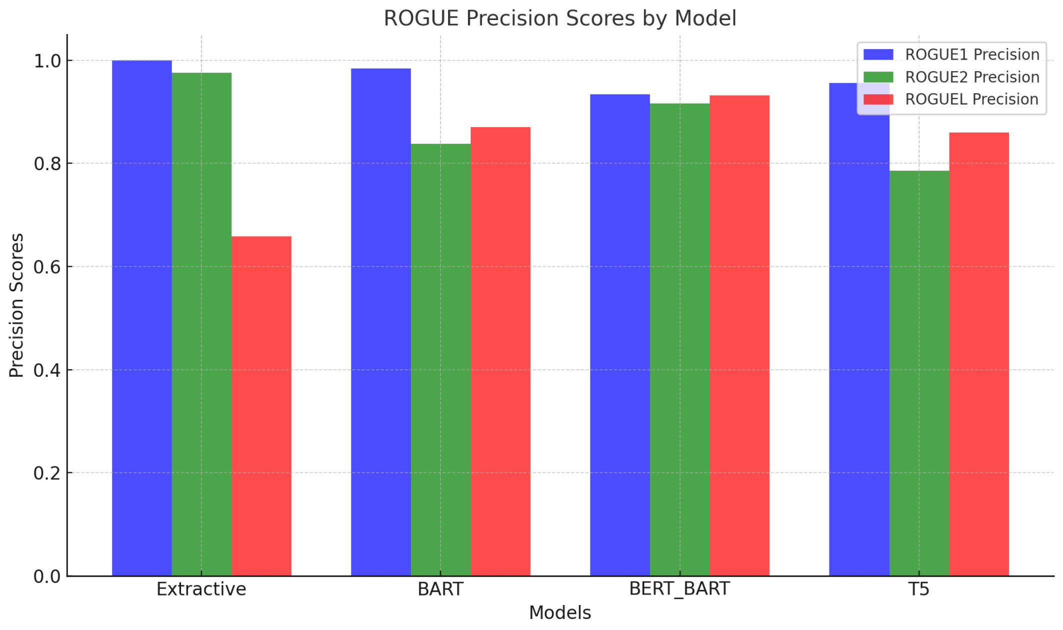


Figure 2e. Precision Comparison of ROGUE 1, 2, and L Metrics Across All 4 Models

III. Analysis

The precision was higher than the recall in both the ROGUE and BERT score. This would indicate that there is accurate information in the summaries but it might be void of some of the crucial concepts found in the original text. When examining the BERT score for the models, the BERT-BART model performed slightly better in all categories except the the extractive model. The extractive model only slightly outperformed the BERT-BART model in both F1 and recall by, 0.02-0.03 respectively. The T5 model performed the worst when it came to BERT score by either being the lowest or tying another model in all metrics. With the values in the BERT score being close to each other across the models shows that the abstractive models are able to create summaries that have information found in the original text. The BERT-BART model seems to have the most accurate and vital information in it's summaries. When looking at the ROUGE scores the extractive model performed exceptionally well when it came to n-gram ROUGE. This is to be expected since the extractive is merely copying the text verbatim. The Rouge-L seems to be an anomaly since a verbatim sentence should also match the longest sequence. The other ROUGE metrics, F1 and recall, being half to a third the amount of the precision shows that either ROUGE is poor at evaluating more complex metrics that allow the developer to understand how capable a model can summarize a text.

IV. Conclusion

Overall, our research was aimed at the differences between various natural language models and how they performed at text summarization. We looked at various models as well as the difference between extractive and abstractive text summarization. We mainly looked at the BART, T5 model, and a combination of the BERT-BART model and their respective results. We gauged the results through Rouge Scores and the BertScore metrics.

After comparing the results, the BERT-BART model scored higher in almost every metric compared to the BART and T5 models. The size of the T5 model might contribute to its low performance since the model used had half the parameters of the other abstractive models. This shows that the added tokenization from the BERT tokenizer can produce far more accurate abstract summaries. In addition, the BERT-BART model had numbers that were very similar to the Extractive model, which scored the highest in almost every metric compared to every other model. We concluded this is because the BERT model almost produces a rough draft of the text summary and the BART model produces a completed version which results in the metrics being so high. This is mainly due to the denoising process of the BART model mixed with the bi-directionality of the BERT model which combined produced the very accurate abstract text summarizations.

In addition, the extractive summary model outscored every other model in both the ROUGE scores and the BERT SCORES, this is beside the ROUGE - L metric. This is most likely because the extractive model pulls words and sentences directly from the given text, and as a result, there is a much higher overlap of words between the summary and the reference text. Due to this, the metric scores are naturally much higher as word overlap is what they are mainly looking for. In addition, since the extractive method pulls from the text the contextuality will be guaranteed to be high.