Text Summarization with Abstractive and Extractive Approaches

Shivani Verma, Nisarg Shah, Suyash Bhagwat, Connor Brown
shivaniverma@umass.edu, nisargshah@umass.edu,
sbhagwat@umass.edu, connorbrown@umass.edu

1 Problem statement

Reading articles and papers can be time-consuming. As a result, people often resort to summaries to get the main gist of a long piece of text. Taking these long documents and extracting the key details from them is a classic, useful task in natural language processing. In this project, we fine-tune the T5 encoder-decoder language model and apply different decoding algorithms to it to evaluate the impact it has on the summaries outputted by the model. We start off with two simple baselines: an extractive approach and an abstractive approach. The main difference between these two approaches is that extractive summaries directly extract sentences from the main document and create a summary out of them. In abstractive summaries, the topic and ideas of the document are maintained in the summary, but the sentences are taken not directly from the document to generate the summary. We use three types of decoding algorithms: beam search, top-k sampling, and nucleus sampling. In addition to this, we also use data augmentation, specifically synonym replacement as a method to increase the model performance in order to further increase the ROUGE for some of the test cases. By applying these techniques to our finetuned T5 model, we hope to produce faithful, coherent summaries with high ROUGE scores.

2 What you proposed vs. what you accomplished

- ~~Collect and preprocess dataset~~: We collected the data from Huggingface, but did not preprocess it as these datasets were already in the format that we wanted and were split into training/validation/test sets.
- Use ChatGPT API as baseline: We did not use the ChatGPT API as our baseline. Upon discussion with Professor Iyyer and TA Yapei, we instead opted for the TextRank model and un-finetuned T5 model as our extractive and abstractive baselines, because ChatGPT is more of an evaluation metric that is expected to outperform our model.
- ~~Build and train an abstractive model and examine its performance~~: We built and trained both an abstractive model and an extractive model, and produced ROUGE scores for both.
- ~~Perform in-depth error analysis to determine what kinds of examples our approach struggles with~~: For our failed examples, we use data augmentation by applying the synonym replacement technique.
- ~~Get human feedback on summaries to see whether the model-generated summaries seem human-like~~: We created a survey to score the model generated summaries against a human written summary to evaluate performance.

3 Related work

The landscape of natural language processing (NLP) and text summarization is wide and diverse, with multiple powerful language models and techniques in use. Research conducted by Ranganathan and Abuka (14) has shown that the T5 model yields state-of-the-art results on various tasks, including text summarization, with performance contingent on the task-specific hyperparameters used. Abdel-Salam and Rafea (1) provide a comprehensive analysis of BERT's use for extractive text summarization, demonstrating its effectiveness across various datasets and tasks. It's

worth noting that the optimal use of BERT, similar to T5, depends on the specific task at hand.

In parallel, (Mihalcea and Tarau) shows that graph-based ranking models have proven effective in text processing. TextRank, for instance, creates a graph where vertices represent words and edges represent co-occurrence relationships between words. The PageRank algorithm is then applied to this graph to rank the words based on their importance. Beam search, another method explored by Chen et al. (4), has been shown to be effective in generating high-quality summaries when used with a pre-trained BERT model.

An alternative approach presented by Bogren et al. (3) involves finding the top-k most similar document pairs and using them to create a summary. This approach has proven its ability to produce accurate and informative summaries across various corpora. Another technique, nucleus sampling, was proposed by (Holtzman et al.) and has been effective in generating more coherent and informative text than beam search, showcasing its potential in various applications such as machine translation, text summarization, and creative writing.

Model performance is often evaluated using ROUGE, a method that measures the overlap between a generated summary and a reference summary. An example of its use can be seen in research from (Kryściński et al.), where it was shown that their model was able to generate more abstract and informative summaries than other models. Moreover, an iterative data augmentation approach for abstract text summarization, which combines paraphrasing, sentence shuffling, and word deletion to create new data points, has proven effective on a German summarization dataset (13).

In a study by Fucci et. al (5), human evaluation was utilized to refine the performance of automatic text summarization systems. The results were used to identify key features and common errors, aiding in system improvement. (Koupaee and Wang) introduced a dataset of 230,000 article and summary pairs from WikiHow to evaluate text summarization methods, showcasing the diversity in topics and methods, from extractive to abstractive summarization. This diversity provides a valuable resource for training and evaluating summarization systems, further fueling advancements in the fields of NLP and text summarization (9).

4 Your dataset

We use two datasets in this project, the CNN/DailyMail dataset (11) and the XSUM dataset (12). The first one is for our extractive baseline approach. The XSUM dataset is used for the abstractive baseline and the main implementation.

For the extractive approach, we use the CNN/DailyMail dataset from Huggingface. This dataset is designed for the text summarization task as it contains articles and their respective highlights. It consists of over 300,000 news articles written at CNN and the Daily Mail. It is split into $287,113$ training, $13,368$ validation, and $11,490$ test examples. Each instance of the data contains a string ID, a string article, and a string highlight which is written by the author of the article. We use version 3.0.0 of the dataset as this is specifically meant for summarization.

For the abstractive approach, we use the XSUM dataset from Huggingface. The data in this set is designed in a similar way to the CNN/DailyMail one–each instance consists of an ID, the document text, and the summary (which is one sentence). The dataset consists of roughly $226,000$ examples, with the following split: $204,045$ training examples, $11,332$ validation examples, and $11,334$ test examples. We use this dataset for the abstractive summarization task because it is known for evaluating single-document abstractive summarization.

4.1 Data preprocessing

For the XSUM and CNN/Dailymail datasets, no preprocessing was required for because they were already in the proper format when retrieved from Huggingface. Since the datasets were split into training/validation/test by default thru the Huggingface API, we did not need to perform the split ourselves.

4.2 Data annotation

We do not use annotations in this project.

5 Baselines

We have two baselines: an extractive and an abstractive baseline. For the extractive baseline, we use the TextRank model, which picks out the most important sentences from the document and then

combines them into a summary. For the abstractive baseline, we use the un-finetuned T5 (T5-base) model from the Huggingface library, which produces a concise summary that is generated by the model. The main difference here is that the abstractive summarization is more challenging than extractive summarization since the model has to understand the entire document and then write the summary instead of directly picking sentences from the document.

5.1 TextRank: Extractive Baseline

TextRank is a model which applies the same graph-based algorithm as PageRank to extract keywords and sentences. In TextRank, a graph is constructed, where the vertices represent each sentence within a document, and the edges between two vertices represents content overlap between the two sentences. Since TextRank gives more importance to words that appear more times in a document, we predict that that the output will not be too faithful to the document, because it may omit key words that are important for the context, but don't occur often. Thus, we include TextRank as an extractive baseline algorithm. We use the `summa` package produced by Barrios et al. (2). More specifically, we use the `summarizer` function, which is used to perform text summarization.

In some cases, the TextRank baseline implementation does not produce a summary at all. We consider these scenarios as failures. After running TextRank on our CNN/DailyMail dataset, we achieved a 0.22% failure rate, where our test set consisted of 11,490 examples.

We calculated the ROUGE scores for our TextRank baseline, as seen in the table below:

-	ROUGE-1	ROUGE-2	ROUGE-L
r	17.0%	0.97%	15.91%
p	7.61%	0.34%	7.07%
f	9.64%	0.46%	8.98%

Table 1: Recall(r), Precision(p) and F-score(f) for the TextRank baseline model

The scores for the TextRank model are not good. To see if the abstractive approach performs better, we add a second baseline: the un-finetuned T5 model, which we discuss in detail in section 5.2.

5.2 Base T5 Model: Abstractive Baseline

The T5 model is an encoder-decoder model which is well-suited for tasks such as text summarization. We use the pre-trained, un-finetuned T5 model (T5-base) from Huggingface as our abstractive baseline to evaluate the quality of abstractive summaries it generates for the XSUM dataset. We select this model because it is an encoder-decoder, so it can generate summaries which maintain the meaning of the original text without extracting sentences verbatim from it.

We then import the `T5Tokenizer` and `T5ForConditionalGeneration` from the transformers library. We pass the XSUM test set, stored in `test_xsum` to the tokenizer for encoding. Once tokenized, the model's `generate()` method is called, which handles "feeding the encoded hidden states" through the cross-attention layers.

As mentioned in section 4.1, our dataset has already been split by default from Huggingface. To get the ROUGE scores for the base T5 model, we use the XSUM test dataset. The values for the ROUGE-1, ROUGE-2, ROUGE-L scores are given in Table 1 below.

-	ROUGE-1	ROUGE-2	ROUGE-L
r	24.13%	3.98%	20.85%
p	15.16%	2.37%	13.27%
f	17.99%	2.84%	15.65%

Table 2: Recall(r), Precision(p) and F-score(f) for base-T5 model

As we can see from the table above, the ROUGE scores on the base-T5 aren't that good. So in order to increase the ROUGE we finetune the T5 model on the XSUM training dataset in the next section.

6 Your approach

To increase the model performance, we now finetune the T5-base model on the XSUM training dataset. The combination of T5 and XSUM is widely popular in the NLP community for text summarization and there is a lot of prior literature available online. To select the optimum hyperparameters for finetuning, we use the hyperparameters published by Ranganathan and Abuka (14). The hyperparameters have been summarized in the Table 3 given below.

Finetuning the entire training data on the T5 model took around 8 hours using Colab Pro.

Hyperparameters	Value
Optimizer	AdamW
Evaluation Strategy	Epoch
Learning Rate	5e-4
Batch Size	4
Weight Decay	0.01
Num of Epochs	5

Table 3: Hyperparameters used for finetuning the T5 model

After finetuning was complete, the finetuned T5 model was used to generate the summaries using three different decoding algorithms; Beam Search, Top-k sampling and Nucleus sampling (also called as top-p sampling). The aim was to select decoding algorithm that generated the best and most coherent summaries. One thing to note here, is that the decoding algorithm that generates the best ROUGE score isn't always the best one. This is because higher ROUGE scores usually indicate repetitive and monotonous text. Hence for the selection criteria, we want a decoding algorithm that achieves a tradeoff between high ROUGE scores and diversity of output text. For all the models given below, we used a static temperature of 1 so that the generated output text isn't overly deterministic. Also, since the f-score is the harmonic mean of the recall (r) and precision (p), we will be using the f-score for all the future comparisons.

Beam Search was the least efficient out of all the three decoding algorithm. This is due to the fact that as we increase the number of beams, the number of hypotheses increases exponentially. In fact, even with Colab Pro(Nvidia A100 GPU) we were unable to run beam search with beam size greater than three due to GPU constraints. The ROUGE scores for beam search are given in Table 4 and are visualized in the Fig 1.

Beam#	ROUGE-1	ROUGE-2	ROUGE-L
1	31.87%	12.23%	26.20%
2	32.27%	12.76%	26.65%
3	32.41%	12.61%	26.12%

Table 4: ROUGE f-scores for Beam Search

Next, we evaluated the performance of top-k sampling. Top-k sampling is much more efficient

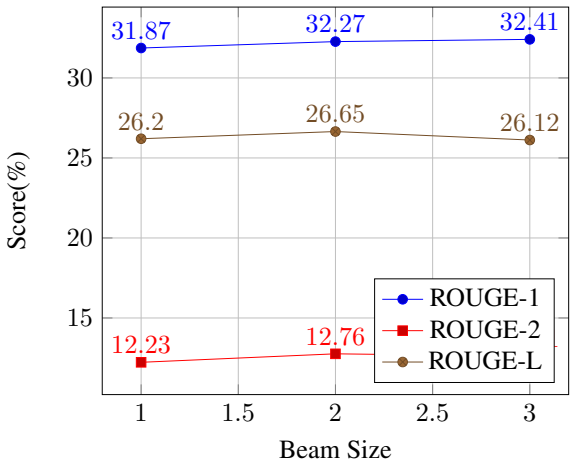


Figure 1: F-Score(%) vs Beam Size

than beam search because instead of having multiple hypotheses, we samples the top k most probable words from the probability distribution. But due to the limitations of Colab Pro, we were only able to sample till k = 7. Above that value, we were getting the 'CUDA Out of Memory' error. The ROUGE scores for top-k sampling and the scatterplot are given in Table 5 and Figure 2 respectively.

k	ROUGE-1	ROUGE-2	ROUGE-L
1	32.32%	11.40%	26.67%
2	29.67%	9.06%	24.08%
3	28.91%	9.04%	23.66%
4	29.14%	8.96%	23.58%
5	28.47%	8.74%	22.93%
6	28.12%	8.40%	22.79%
7	27.58%	8.43%	22.57%

Table 5: ROUGE F-scores for Top-k sampling

Lastly, we used nucleus sampling with value of p ranging from 0.7 - 1.0. The ROUGE scores for nucleus sampling are given in Table 6 and are visualized in the Fig 3.

We can compare all three decoding algorithms side by side in Fig 4.

Also, Fig 7 compares the ROUGE score of the two baselines along with the T5-finetuned model and we did see an improvement in the ROUGE score post finetuning. Post finetuning, a few examples of the test cases with a high ROUGE score are given in Fig 6. Also some of the test cases which have a low ROUGE score are given in Fig 5.

But, as mentioned earlier, the selection criteria for the best decoding algorithm doesn't only de-

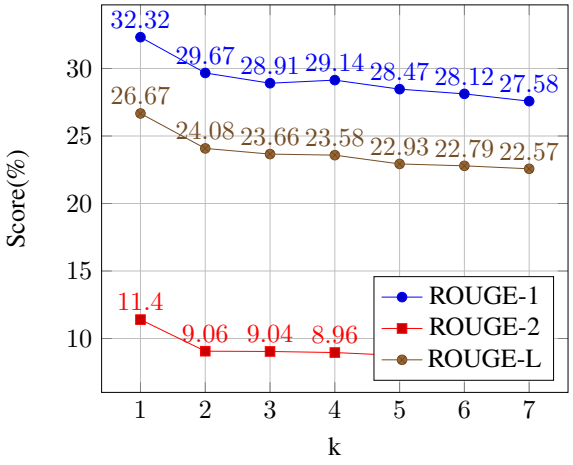


Figure 2: F-Score(%) vs k

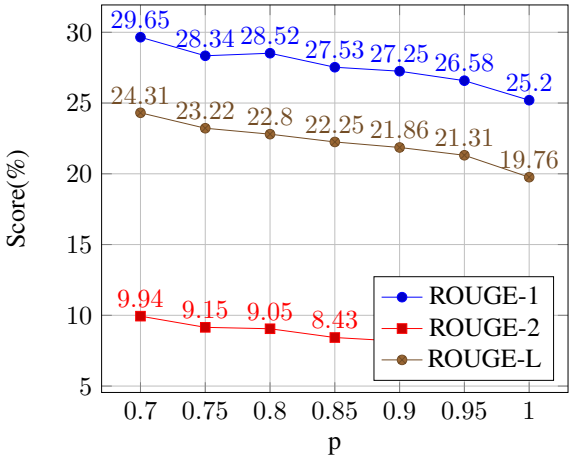


Figure 3: F-Score(%) vs p

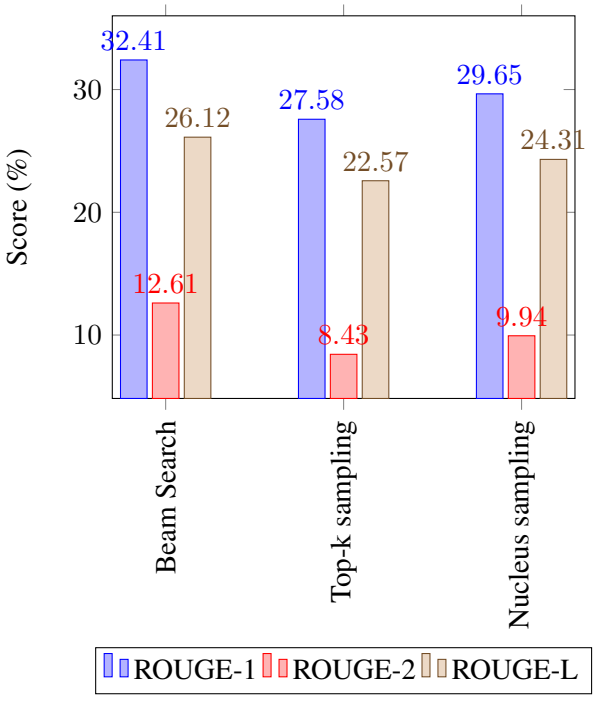


Figure 4: ROUGE f-scores for beam size=3, k=7 and p=0.7.

p	ROUGE-1	ROUGE-2	ROUGE-L
0.7	29.65%	9.94%	24.31%
0.75	28.34%	9.15%	23.22%
0.8	28.52%	9.05%	22.80%
0.85	27.53%	8.43%	22.25%
0.9	27.25%	8.17%	21.86%
0.95	26.58%	7.55%	21.31%
1.00	25.20%	6.83%	19.76%

Table 6: ROUGE scores for Nucleus or Top-p sampling

Model Summary: Police are appealing for witnesses after a man was stabbed near a house in Inverness.
Target summary: Police in Fife are searching for a man who was seen exposing himself in Dunfermline.
Average ROUGE score: 0.3571428521428573

Model Summary: A buzzard has been rescued from a fence in Kent after getting stuck in wire.
Target summary: A buzzard has been rescued after becoming trapped between fencing and a wall in Gwynedd.
Average ROUGE score: 0.4400656764489554

Model Summary: A teenager was knocked down by a car while crossing a road in Rotherham.
Target summary: A 14-year-old boy has been left with "life-threatening" injuries after being knocked down by a car.
Average ROUGE score: 0.3711001592477878

Figure 5: Examples of test cases with low f-score with finetuned T5.

pend on the ROUGE score. We also compared the output summaries generated by all three algorithm in the 'Visualizing decoding algorithms' section in the ipynb notebook and checked the coherency and quality of the output. By looking at the generated output, nucleus sampling with p = 0.7 and temperature = 1.0 gives us the most coherent output text. This result is confirmed by (Holtzman et al.) as well. Hence, in order to have a good balance between the ROUGE scores and coherency of the generated summaries, we will be using nucleus sampling with p = 0.7 as the main decoding algorithm for all future sections of this report.

Original Summary 1: There is a "chronic" need for more housing for prison leavers in Wales, according to a charity.

T5 Summary 1: prison link cymru says some ex-offenders were living rough for up to a year. charity says investment in housing would be cheaper than jailing repeat offenders. spokesman says national pathway for homeless services has prevented many losing home.

Original Summary 2: A man has appeared in court after firearms, ammunition and cash were seized by police in Edinburgh.

T5 Summary 2: officers recovered three firearms, ammunition and a five-figure sum of money. A 26-year-old man was arrested and charged on Thursday.

Figure 6: Examples of test cases with high f-score with finetuned T5.

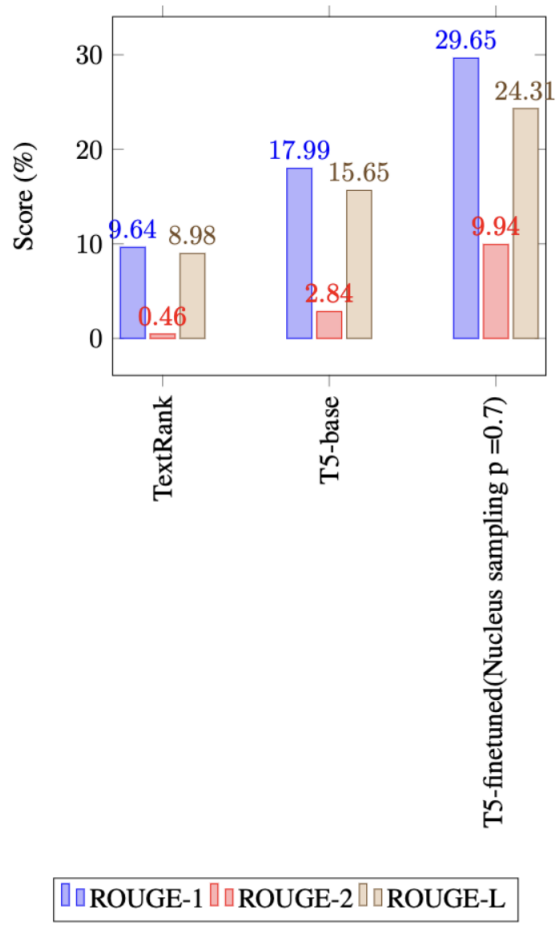


Figure 7: ROUGE F-scores for different models

7 Error analysis

7.1 TextRank

The first errors we encountered were encountered whilst analyzing the performance of our extractive baseline, using an implementation of TextRank. The most notable flaw was a situation in which some of our CNN/Dailymail test cases actually produced null, or simply empty, summaries. These cases were quite odd; a few examples of these are shown in Figure 8.

Namely, we hypothesized that the problem may be caused by characters that the implementation could not process. For example, it can be seen in "Original Article 294" that bullet characters appear, implying that their presence could be confusing as a potential delimiter. However, this hypothesis is not defended by the second example, "Original Article 9566," which contains a perfectly normal article that still produces a null summary. Thus, we could not form a proper hypothesis for the cause of our extractive baseline's errors.

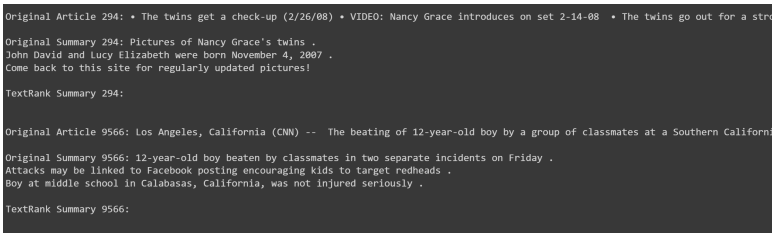


Figure 8: Examples of error cases within TextRank. **Original Articles** are chunks of the articles to summarize, **Original Summaries** are the label sum maries, and **TextRank Summaries** are the empty outputs.

Overall, our TextRank implementation processed 11,490 articles, with 26 of those articles being "failures," producing these empty summaries. This left us with a **0.2263%** rate of failure when producing summaries for CNN/Dailymail news articles. For the sake of keeping good time on other portions of our project and the incredibly low rate of this error occurring, we chose to simply void these 26 error cases and utilize the results we obtained from the remaining 11,474 cases.

7.2 T5 Error Analysis

Based on the results of our analysis on 1% of the XSUM dataset (results in Section 6), we chose to continue the remainder of our analysis of our fine-tuned T5 model utilizing nucleus sampling with p of 0.7. For our error analysis, we worked to find a method of improving ROUGE scores further by analyzing test cases that did not contribute well to our final, averaged scores. We settled on augmentation, specifically replacing words within the test cases with synonyms, to see if the fault was due to randomness, i.e. will subtly altering worse test cases improve results, or if our model is simply unable to capture the context of these articles well.

The process followed first choosing a threshold for "bad" test cases, which we chose to be any test case that produces an average F1-score (averaged between ROUGE-1, ROUGE-2, and ROUGE-l) that is approximately, less than half of the average F1-SCORE without augmentation using Top-p with p = 0.7, which was 21.56% (averaged from Table 4). From the 1% we processed (due to aforementioned constraints), we found a total of 19 test cases that we could augment to hopefully improve scores. We then pipelined these test cases into a text augmentation implementation produced, proposed, and made publicly available by Marivate and Sefara (9) in a Python pack-

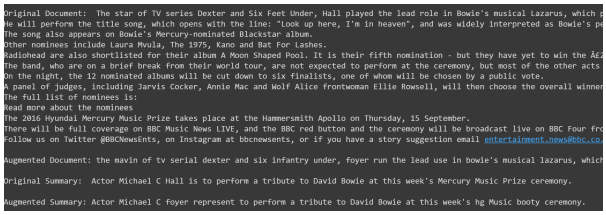


Figure 9: Example of an augmented test case, showing the original article, augmented article, the original label summary and the augmented label summary. Note the word changes between originals and their respective augments.

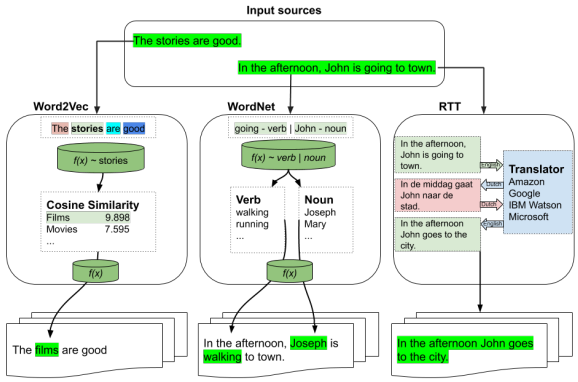


Figure 10: A visual representation of the textaugment package's usage.

age called `textaugment`. Their proposal involved using other word models to augment text based on random selections from probability distributions of synonyms for words through a variety of libraries, including Word2vec and Wordnet, the latter of which we chose to use. Using Wordnet, we augmented not only the articles that we found needing augmenting, but their associated summaries as well, replacing words in the summary that were replaced in the article. An example of this is shown in Figure 9. A visual representation of the process they created is also shown in Figure 10.

Unfortunately, this process seemed to not be a method that fit all situations. The process of augmentation actually *reduced* our ROUGE scores, shown in Table 7. Despite the decrease, we can show that this process does have some benefit. Post-augmentation, we can look at what cases of our new data were instead *above* our threshold for augmentation, and see if any of those were among the cases we chose to augment previously. These overlapping cases would theoretically represent cases that were augmented and had their contributions improved. For our 19 augmented test cases in XSUM, 21.05% of them actually had their contributions to the ROUGE scores improved by the augmentation process. Thus, while our ROUGE scores did decrease, with some improvement or better application of the augmentation, we can still see some individual improvements.

Time	ROUGE-1	ROUGE-2	ROUGE-L
Before	29.65%	9.94%	24.31%
After	29.41%	5.88%	17.65%

Table 7: ROUGE scores for XSUM before and after augmentation

8 Human Evaluation

8.1 Overview

Since the goal of our project is to generate accurate, coherent, and fluent summaries that are human-like, we construct a survey for humans to fill out. In our survey, we give a brief one sentence idea of what a document is about. Then, we present the participant with two summaries for the document and ask them to pick which one sounds better to them. Each question follows this format and has three multiple choice options: the ground-truth summary, the model generated summary, and "Not sure". Note that this is "blind evaluation", meaning that the participants who take the survey do not know which summary is model generated.

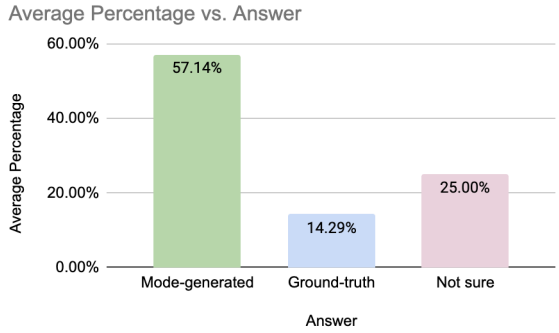
With the survey, we hope to see participants select the model generated summary. If they do, this means that our model generated summaries are fluent and coherent enough and maintain the context of the document.

The purpose of including the option "Not sure" for each question is that it will inform us of when participants are having a hard time selecting one summary over another. In a way, selecting this option is good because it means that our summary was not incoherent/ungrammatical, because the participant would have picked the other summary in that case. If they select "Not sure", that means that the model generated summary is at the same (or similar) level of sophistication that as the ground truth summary.

8.2 Survey Results

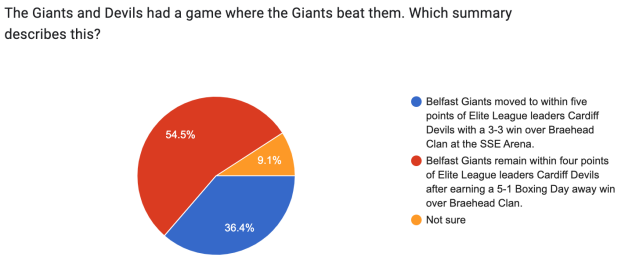
On average, the model generated summary was selected 57% of the time, while the ground truth summary was only selected 14% of the

time. Interestingly, the "Not sure" option was selected on an average of 25% of the time. The below chart shows the average amount of time each type of answer was selected.



Assuming that participants faithfully answered the questions, the survey indicates that the model generated summaries are coherent and fluent. They also appear to be faithful to the meaning of the document based on the sentence that was written in the question.

One question on the survey had the highest number of model generated answer selected along all questions. The below pie chart shows how the answers were spread out.



In the above pie chart, the red section represents the model generated summary while the blue represents the ground truth summary. As seen in the legend, the two summaries do not vary much, so it is understandable if participants select the model generated one.

9 Conclusion

Text summarization is a useful feature in many, if not all fields. Whether doctors want to summarize their patient's visit summary or students want to summarize articles, using natural language process to complete this task increases efficiency. Our model generated summaries have shown us that abstractive methods result in more natural, human-like summaries as compared to extractive methods. The data augmentation part of the project was slightly challenging initially, because it required replacing words within articles and summaries with their synonyms, which was challenging and tedious. To our surprise, augmentation reduced ROUGE scores in some cases. We expected augmentation to increase ROUGE scores by allowing the model to better generalize. However, many cases did have better ROUGE scores, as we expected. As a future extension, it would be interesting to try different augmentation techniques and compare them to the synonym replacement to see if they increase our model's performance. Furthermore, it would be interesting to test other datasets that are unrelated to the news, such as academic papers, on our model. Since different types of writing have different styles, formats, and level of sophistication, the model may perform better or worse on them. All in all, summarization is an important task that is useful in many areas, and our findings in this project have provided us with a deeper understanding of how models encode and decode text.

10 Contributions of group members

List what each member of the group contributed to this project here. For example:

- **Connor Brown:** Implemented and analyzed TextRank extractive baseline, performed error analysis; Performed Error analysis of Finetuned T5 on XSUM; Analysis and results of un-finetuned T5 abstractive baseline; Additional work on related works and research

- **Suyash Bhagwat:** Performed finetuning of the base-T5 model on XSUM dataset; Decoding algorithm implementation(Beam Search, Top-k and Nucleus Sampling); Histograms, scatterplots and tables in the Approach section; Code for the Human Evaluation section

- **Shivani Verma:** Setting up the boiler plate code in the Colab notebook, adding the CNN/Dailymail dataset, researching and sharing data augmentation libraries with the team, creating scatter plots for the Approach section of the report, implementing sections 1, 2, 4, 5, 8, and 9 of the report, and creating the human evaluation survey to send to people.

- **Nisarg Shah:** Worked on researching and writing for the Related Works section of the final report, assisted in analysis of un-finetuned T5 model. Worked on the What

you proposed vs what you accomplished section.

11 AI Disclosure

- Did you use any AI assistance to complete this proposal? If so, please also specify what AI you used.
  - Yes, we used **chatGPT** and **GPT4**.

If you answered yes to the above question, please complete the following as well:

- T5 Finetuning – Prompt:Write a PyTorch program that loads a fine-tuned T5 model from a folder named 't5-finetuned'.
- T5 finetuning – Prompt: Write a Python program using PyTorch to fine-tune a T5 model on the training dataset. Assume that the training dataset is already available to us as a torch tensor.
- Beam Search – Prompt: Write a PyTorch function that loads a saved T5 model, uses the beam search decoding algorithm to produce outputs and then compares the ROUGE scores for the output produced with the target text.
- Top-k sampling – Prompt: Can you update the above code to use top-k sampling instead of beam search as the decoding algorithm?
- Nucleus sampling – Prompt: Can you update the above code to use top-p sampling instead of top-k sampling as the decoding algorithm?
- Nucleus sampling – Prompt: What is a good estimate for the value of p in the above code?
- Data Augmentation – Prompt: Can you modify the above code to print out all the (summaries, target_summaries) pairs that are below a certain threshold 't'?
- Data Augmentation – Prompt: Can you update the above code to also print out the input_texts corresponding to the summary?
- Generating tables in LaTex – Prompt: Write program in LaTex to generate a 5 x 4 table.
- Generating histogram in LaTex – Prompt: Can you write a program in LaTex to add a multiple entry histogram with 4 categorical variables on the X-axis (3 entries per variable) and a continuous Y-axis?
- Adding screenshots to LaTex – Prompt: Write a LaTex program to add a screenshot of an image to the LaTex document.
- Generating scatterplots in LaTex – Prompt: Write a LaTex program to add a scatterplot to the Latex document which includes three different lines in the same plot.
- Combine and make a rough draft of all the related works paragraph below into 1 section.
- **Free response:** For each section or paragraph for which you used assistance, describe your overall experience with the AI. How helpful was it? Did it just directly give you a good output, or did you have to edit it? Was its output ever obviously wrong or irrelevant? Did you use it to generate new text, check your own ideas, or rewrite text?
  - T5 finetuning - The output produced by GPT4 was somewhat accurate. It was able to get the basic template correct, but there were minor issues that had to fixed. We had to edit the GPT4 code and then it was able to run on Colab.
  - Beam Search - The output produced by GPT4 was accurate. The code was concise and error-free.
  - Top-k sampling - The output produced by GPT4 was somwhat accurate. There were minor errors that had to be corrected.
  - Nucleus sampling - The output produced by GPT4 was somwhat accurate. There were minor errors that had to be corrected.
  - Data Augmentation - The output produced by GPT4 was reasonably accurate. Apart from a few minor corrections, the code was correct.
  - Tables, Histogram,Plots,Screenshot - The output produced by GPT4 was accurate.

12 Acknowledgements

Professor Iyyer and TA Yapei Chang provided us with helpful explanations and feedback during their office hours. They guided us through the

project and gave us great suggestions which we incorporated. We are thankful for all the help we recieved from them.

References

[1] Abdel-Salam, S. and Rafea, A. (2022). Performance study on extractive text summarization using bert models. *Information*, 13:67.

[2] Barrios, F., López, F., Argerich, L., and Wachenchauzer, R. (2016). Variations of the similarity function of textrank for automated summarization. *CoRR*, abs/1602.03606.

[3] Bogren, E. and Toft, J. (2014). Finding top-k similar document pairs -speeding up a multi-document summarization approach.

[4] Chen, X., Li, J., and Wang, H. (2019). Keyphrase guided beam search for neural abstractive text summarization. *IEEE Xplore*, page 1–9.

[5] Fucci, D., Romano, S., Baldassarre, M., Caivano, D., Scanniello, G., Thuran, B., and Juristo, N. (2022). A longitudinal cohort study on the retainment of test-driven development.

[Holtzman et al.] Holtzman, A., Buys, J., Li, Forbes, M., Choi, Y., and Allen, P. The curious case of neural text degeneration.

[Koupaee and Wang] Koupaee, M. and Wang, W. Wikihow: A large scale text summarization dataset.

[Kryściński et al.] Kryściński, W., Paulus, R., Research, S., Xiong, C., and Socher, R. Improving abstraction in text summarization.

[9] Marivate, V. and Sefara, T. (2020). Improving short text classification through global augmentation methods.

[Mihalcea and Tarau] Mihalcea, R. and Tarau, P. Textrank: Bringing order into texts.

[11] Nallapati, R., Xiang, B., and Zhou, B. (2016). Sequence-to-sequence rnns for text summarization. *CoRR*, abs/1602.06023.

[12] Narayan, S., Cohen, S. B., and Lapata, M. (2018). Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *CoRR*, abs/1808.08745.

[13] Parida, S. and Motlicek, P. (2019). Abstract text summarization: A low resource challenge. page 5994–5998.

[14] Ranganathan, J. and Abuka, G. (2022). Text summarization using transformer model. page 1–5.