News Summarization using Enhanced Content Features

1st Andreas Loutzidis
Computer Science
Columbia University
New York, USA
al4419@columbia.edu

2nd Rajwinder Mahal
Computer Science
Columbia University
New York, USA
rsm2207@columbia.edu

***Abstract*—Online news reading has become one of the most popular ways to consume the latest news. News aggregation websites such as Google News and Yahoo News have made it easy for users to find the latest news and provide thousands of news stories from hundreds of news publishers. As people have limited time, reading all news articles is not feasible. As the success of zero-shot and few-shot prompting with models like GPT-3 has led to a paradigm shift in NLP research, we conducted experiments to explore the feasibility of using enhanced content features such as news categories and named entities to improve the quality and coherence of generated news summaries. Our results provide insights into the effectiveness of different LLMs (BART and PEGASUS) for news summarization and the impact of additional content features such as topic modeling, named entities, and sentiment analysis. The proposed approach could be beneficial for users who consume news online and want to quickly and efficiently obtain the most relevant information.**

***Index Terms*—news aggregation websites, zero-shot prompting, few-shot prompting, GPT-3, NLP research, enhanced content features, news categories, named entities, LLMs, BART, PEGASUS, news summarization, topic modeling, sentiment analysis**

I. Introduction

Online news reading has become one of the most popular ways for people to stay informed about the latest events around the world. With the advent of news aggregation websites such as Google News and Yahoo News, it is easier than ever to access news all around the world. However, with the amount of information that is available, it is not feasible to consume every article in detail. Although most news aggregators use recommendation systems to show the news based on user preferences, it is still not feasible to consume all the information. As a result, there is a growing need for efficient and effective methods to make it easier to consume information at scale and help readers quickly consume the most important information.

In recent years, there has been increasing interest in using natural language processing (NLP) techniques for news summarization. More specifically, large language models (LLMs) such as GPT-3 [1] have shown impressive performance in a wide range of NLP tasks including machine translation, question-answering, and text generation. As a result, there has been increasing interest in using LLMs for news summarization.

However, these existing methods have limitations, including low-quality summaries and computational complexity, which make them unsuitable for practical applications at scale. To address these issues, we experimented with a new approach that uses additional content features such as news headlines, categories, topic modeling, and named entities to improve the quality and coherence of generated summaries. We also experimented with different optimization techniques such as quantization, model pruning, and distributed training to reduce the model size and improve both training and inference time. Our goal is to achieve state-of-the-art performance on news summarization benchmark datasets while keeping the model as lightweight as possible.

In this paper, we present the design and implementation details of our proposed approach and its performance evaluation. Specifically, we fine-tuned BART and PEGASUS on CNN/DailyMail dataset [2] and examined their performance.

II. Related Work

News summarization is one of the most important tasks in the field of natural language processing. In the early days, news summarization used heuristics-based methods such as sentence extraction and sentence compression to generate summaries. As these methods failed to capture the underlying meaning of the text, they often produce low-quality results. In recent years, there are new approaches that use machine learning techniques to improve the quality of generated summaries. These approaches include both extractive and abstractive summarization methods. In extractive summarization, the goal is to select a subset of sentences from the input text to create a summary. A common approach is to use supervised learning methods to rank sentences based on their importance to the overall meaning of the article. This approach has shown promising results in generating high-quality summaries but it requires a large amount of annotated data. In abstractive summarization, the goal is to generate a new text that summarizes the input text. It often generates summaries that are more coherent and informative than extractive summarization, however, these abstractive summarization methods are more challenging to implement and require more sophisticated machine learning techniques such as deep neural networks.

In recent years, there has been increasing interest in using large language models (LLMs) such as GPT-3 for news summarization. These models have shown state-of-the-art results

on a wide range of NLP tasks including news summarization. However, these models still have some limitations in terms of summarization quality and computational complexity. To address these limitations, we are conducting various experiments to improve news summarization and develop a lightweight model that is computationally less expensive but still generates high-quality summaries.

III. Models and Datasets

The central components of our approach are the state-of-the-art pre-trained models that we are going to leverage and fine-tune to achieve our goals.

A. BART - Bidirectional and Auto-Regressive Transformers

Bidirectional and Auto-Regressive Transformers (BART) [3] is a sequence-to-sequence transformer model developed by Facebook AI Research (FAIR). It is based on the transformer architecture introduced in the original paper by Vaswani et al. (2017) [4] and uses a combination of denoising auto-encoding and back-translation techniques. During pre-training, it uses a noise function to corrupt the text data so that the model can learn to reconstruct the original text. It utilizes a standard sequence-to-sequence transformer architecture except that it uses GELU activation functions instead of ReLU. The base model has 6 layers in the encoder and decoder. Each decoder layer performs cross-attention over the final hidden layer of the encoder. In Figure 1, the encoder inputs are corrupted using mask symbols which are then encoded using the bidirectional encoder. Then the likelihood of original inputs is calculated with an autoregressive decoder. This pre-training technique allows the model to learn a more robust understanding of the structure of the input data and has shown excellent results in various tasks such as text generation and summarization. Figure 2 shows that the BART has a total of 560 million trainable parameters.

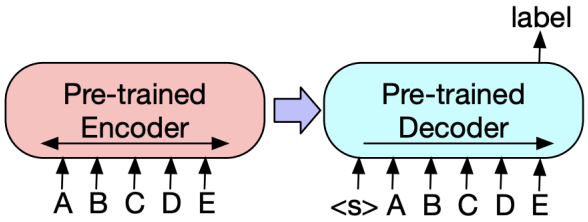

Fig. 1. BART encoder-decoder.

B. PEGASUS - Pre-training with Extracted Gap-sentences for Abstractive Summarization

PEGASUS [5] is a pre-trained large transformer-based encoder-decoder model that has shown impressive results in abstractive summarization. Unlike other pre-training models such as BERT which mask individual words in the input text, PEGASUS masks complete sentences in the input text during pre-training as shown in its architecture in Figure 3. This allows PEGASUS to focus on the most important

```
==========================================================================
Layer (type:depth-idx)                                   Param #
==========================================================================
├─BartForConditionalGeneration: 1-1                      --
|    └─BartModel: 2-1                                    --
|    |    └─Embedding: 3-1                               51,471,360
|    |    └─BartEncoder: 3-2                             203,678,720
|    |    └─BartDecoder: 3-3                             254,084,096
|    └─Linear: 2-2                                       51,471,360
==========================================================================
Total params: 560,705,536
Trainable params: 560,705,536
Non-trainable params: 0
==========================================================================
```

Fig. 2. BART layers and parameters.

parts of the input text and generate more informative and coherent sentences. In our experiments, we used the Hugging Face distribution of PEGASUS-large which has 197 million trainable parameters and 669 million non-trainable parameters, as shown in Figure 4. As PEGASUS uses unsupervised pre-training techniques, it has a large number of non-trainable parameters. These non-trainable parameters are used for various tasks such as embeddings, positional encoding, and normalization, and these parameters are not updated during fine-tuning or downstream tasks. However, PEGASUS still has a large number of trainable parameters that allows it to capture more complex relationships between input and output text during fine-tuning. Overall, PEGASUS has shown state-of-the-art results in text summarization, including news summarization.

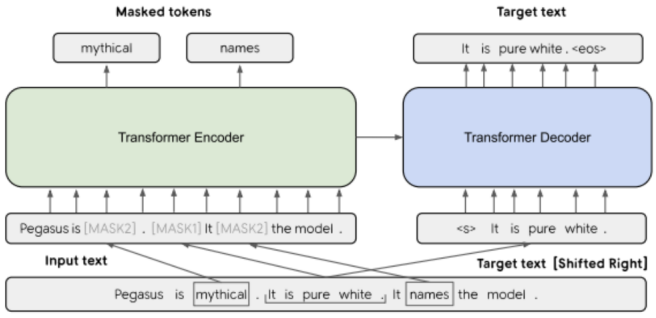

Fig. 3. Pegasus encoder-decoder.

```
==========================================================================
Layer (type:depth-idx)                                   Param #
==========================================================================
├─PegasusForConditionalGeneration: 1-1                   --
|    └─PegasusModel: 2-1                                 --
|    |    └─Embedding: 3-1                               98,409,472
|    |    └─PegasusEncoder: 3-2                          (300,999,680)
|    |    └─PegasusDecoder: 3-3                          (368,206,848)
|    └─Linear: 2-2                                       98,409,472
==========================================================================
Total params: 866,025,472
Trainable params: 196,818,944
Non-trainable params: 669,206,528
==========================================================================
```

Fig. 4. PEGASUS layers and parameters.

C. Daily Mail CNN Dataset

The CNN / Daily Mail dataset is an English-language dataset consisting of just over 300,000 unique news articles

authored by journalists at CNN and the Daily Mail. The dataset was initially created for machine-reading comprehension and abstractive question answering, but it supports both extractive and abstractive summarization tasks as well. Each data instance contains three fields: an identifier ('id'), the full text of the news article ('article'), and a summarized version ('highlights'). For our study, we will be using the 'highlights' field as the target summary of the 'article' field. The dataset displays a token count per instance with an average of 781 tokens for the 'article' field and approximately 56 tokens for the 'highlights' field. This difference underscores the summarization challenge inherent in the dataset, as the model will need to distill the essence of a lengthy article into a much shorter summary. To ensure robust training and accurate evaluation, we will use the recommended split of the dataset into training, testing, and validation subsets. Specifically, there will be approximately 287,000 instances for training, 11,000 for testing, and 13,000 for validation. During the initial stages of the project, we will employ the training and validation subsets to search for optimal hyperparameters. We will then conduct the final evaluation on the test subset, keeping it separate throughout the process to ensure an unbiased evaluation of our model's performance. The decision to use the CNN / Daily Mail dataset for our research is primarily driven by resource constraints and the scope of our project. Despite its limitations, this dataset provides a valuable opportunity to study the performance of our models in the context of news article summarization.

IV. Evaluation Metrics

As we are using sequence-to-sequence models, we used the softmax cross-entropy loss as the primary metric to optimize the model. The softmax cross-entropy measures the difference between the predicted and actual probability distributions of the target tokens in the output sequences. By minimizing the loss, the decoder learns to generate sequences, in our case summaries, that are more similar to the ground truth.

For the evaluation of our fine-tuned models, we will be using the industry-standard Recall-Oriented Understudy for Gisting Evaluation (ROGUE) [6] which measures the performance of automatic summarization and machine translation tasks. The evaluation compares the inference result (summary of a text) of a model against a set of human-generated reference summaries. To evaluate the quality of the summary after combining different results we will be using the Bilingual Evaluation Understudy (BLEU) [7] score which is an algorithm for evaluating precision.

Initially, we planned to use a tiny test set (< 100 articles) to generate summaries using our model as well as using existing models such as GPT-3 and ChatGPT and then use a human to rank summaries without telling which one is generated by which model. Our intuition was that this will help us understand whether our model produces results that are on par with state-of-the-art LLMs such as GPT-3 and ChatGPT which are much larger than our proposed model. However, due to limited time and resources, we couldn't perform the human evaluation of generated summaries and this is something that we will consider in future work.

V. Training Methodology

Training large language models (LLMs) can be a daunting task, particularly when time and resources are limited. Due to the size of these models, it is not practical to train them on a single GPU. Therefore, we utilized distributed training on Google Cloud to accelerate the process. Our training was performed using four Tesla V100 GPUs, implementing a distributed data-parallel strategy to effectively reduce training time. We used virtual machines with 32 vCPUs and at least 120GB RAM.

To streamline the training process and minimize potential issues, we employed PyTorch Lightning, a framework that simplifies the implementation of distributed training. This allowed us to focus on the training strategy itself, while the framework handled the complexities of setting up the distributed training.

To keep track of our training process, we used Tensorboard for logging and Weights & Biases [8] for hyperparameter search. While we initially attempted to use PyTorch Profiler for profiling our models, we encountered memory errors due to the large size of our models. Consequently, we performed some profiling using the built-in time package in Python as an alternative solution. However, we may opt to use PyTorch Profiler if we get access to a better GPU with higher memory, such as the Nvidia A100.

During training, we used machine learning best practices and user training and validation split to track the learning process of our models. We used test split only during the final evaluation of our fine-tuned models.

Throughout the training process, we adhered to machine learning best practices and utilized a training and validation split to monitor our models' learning progress. We reserved the test split exclusively for the final evaluation of our fine-tuned models. We also tried to employ early stopping to prevent overfitting and reduce training time and tried to monitor the validation loss to perform early stopping. However, the training time was taking over 10 hours per epoch so we had to let it run overnights and we couldn't actively monitor the loss during the training although we kept track of all logs using the Tensorboard.

A. Batch Size

The batch size is a hyperparameter that determines the number of training samples used in one iteration of gradient calculation. A larger batch size typically results in faster convergence but requires more memory and computation resources. In our case, we used a batch size of 4 per GPU to maximize GPU memory utilization. We tried to go higher but that resulted in a memory error. In total we used 4 GPUs, resulting in an effective batch size of 16.

The original input and output size of 1024 tokens was initially selected for our experiments, but we faced issues running experiments with this size due to its large memory

footprint on our GPUs. To overcome this issue, we analyzed the training dataset to compute the mean and median length of input text and target summaries. Our analysis revealed that the average article length was approximately 600 words, and the average summary length was around 64 words. Based on these findings, we reduced the input and output sizes to 512 and 128 tokens, respectively. This adjustment allowed us to use a batch size of 4 per GPU, and thus better utilize our available computational resources.

B. Learning Rate

Initially, we set the base learning rate to 5e-5, but we later performed a hyperparameter search to determine the optimal learning rate (see the next section for details). Additionally, we employed a linear learning rate with a warmup scheduler from the Transformers library, which is a widely used technique. The warmup scheduler increases the learning rate during the initial phase of training, followed by a linear decrease in the learning rate as training progresses. We used it to improve the model convergence and prevent the model from getting stuck in the local minima. By gradually increasing the learning rate during the initial warmup phase, the model can explore a larger portion of the parameter space which helps in faster convergence and improved generalization.

VI. Performance Tuning Methodology

Hyperparameter tuning is a critical component of machine learning training. To facilitate this process, we utilized Weights & Biases to perform a random hyperparameter search. However, due to the enormity of our models and dataset, each sweep run was taking roughly 10 hours to complete. To overcome this challenge, we performed a hyperparameter search on a subset of our training data, which accounted for approximately 25% of the entire dataset. Additionally, we employed a validation split to evaluate the model's performance during the hyperparameter search. We conducted ten random sweep runs for each of our models, BART and PEGASUS. Our primary objective in the hyperparameter search was to tune the learning rate, weight decay, and the ratio of warmup steps to the total number of training steps. For the evaluation of our models, we used the ROUGE score to measure the performance of generated summaries and the BLEU score to evaluate the quality of the generated summaries. The results of our hyperparameter search will be presented in the next section.

VII. Experimental Results

In our experiments, we used pre-trained BART large and PEGASUS large. We conducted different experiments including a hyperparameter search. In this section, we will present and discuss the outcomes of our experiments.

In our initial experiments, we conducted a hyperparameter search using Weights & Biases on a small subset of our dataset due to its large size and increased training time. We conducted 10 random sweeps for both BART and PEGASUS. Figure 5 shows the hyperparameter sweeps for PEGASUS. We observed that using warm-up steps with weight decay caused PEGASUS to have a significantly higher loss. As shown in Figure 6, the warm-up step ratio is the most important hyperparameter for PEGAUS. Consequently, we conducted two separate experiments (explained later) for PEGASUS - one with the best warm-up step ratio of 0.05 and weight decay of 0.01, and another experiment without warm-up steps and weight decay. Figure 7 shows the training loss for the BART hyperparameter search. Like PEGASUS, BART exhibits similar behavior concerning warm-up steps as shown in Figure 8, with warm-up step ratio being the most important hyperparameter, but with slightly less relative weight than PEGASUS.

After performing a hyperparameter search, we conducted four experiments: BART with and without warmup steps and PEGASUS with and without warmup steps. Figure 10 displays the validation loss of these experiments. We observed that both BART and PEGASUS have high validation loss with warmup steps, with PEGASUS having a much higher loss than BART. Without warmup steps, both models have very similar validation loss. However, we noticed that the loss does not decrease as we train for more steps. We also evaluated these experiments using ROUGE and BLEU scores. Figure 10 shows the evaluation scores on the test set. We found that PEGASUS without the warmup step performs the best, with BART without the warmup steps following closely behind.

Finally, we conducted experiments to evaluate the performance of these fine-tuned models after quantization and pruning. We used a post-training dynamic quantization approach using the Eager Mode Quantization method in PyTorch. We quantized all four fine-tuned models in the previous experiments. As expected all models performed worse than the original models, as shown in Figure 11. But we noticed that PEGASUS without the warmup steps tend to have slightly better performance than the original model and much better performance than all other models in the experiments.

During the pruning of models, we experimented with various pruning rates ranging from 20% to 80%. As shown in Figure 12, we observed that a pruning rate of 40% gave the best performance for all models except for BART without weight decay, where an 80% pruning rate was better. Moreover, we found that PEGASUS without warmup steps had the best overall performance. Therefore, we conducted an additional experiment of quantization to further investigate its capabilities and its performance in the test dataset. The quantized model, as expected, performed worse in both ROUGE scores (for unigram and bigram) as well as on the BLEU score. However, it was almost 2 times faster in inference and almost 7 times smaller in size. This behavior demonstrates the trade-off between the model size and its ability to perform demanding NLP tasks.

VIII. Challenges

In our endeavor, we faced several significant challenges. Incorporating additional features such as topic modeling, named entities, and news categories necessitated the use of

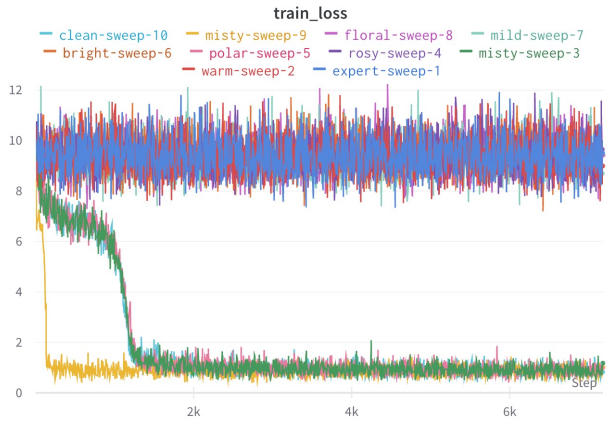

Fig. 5. PEGASUS hyperparameter tuning sweep.

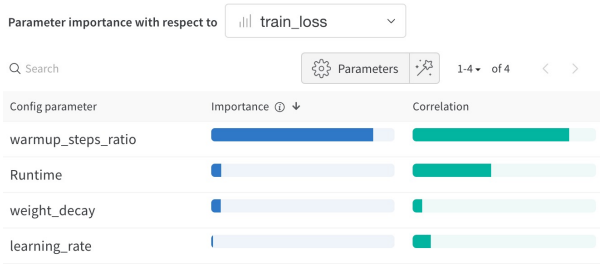

Fig. 6. PEGASUS feature importances.

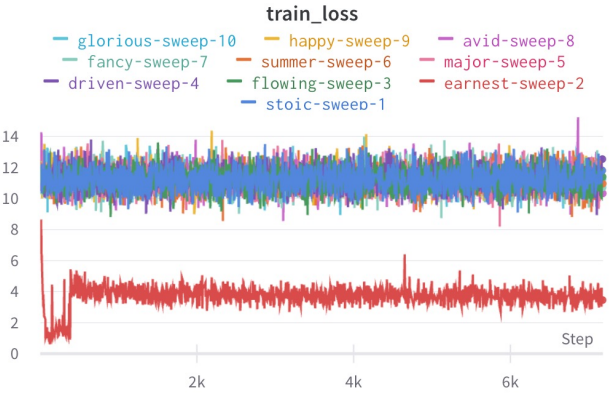

Fig. 7. BART hyperparameter tuning sweep.

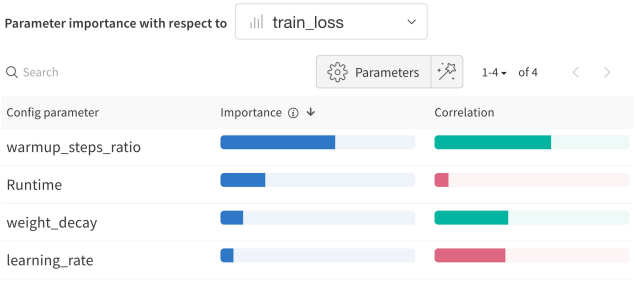

Fig. 8. BART feature importances.

	ROUGE-1	ROUGE-2	BLEU
PEGASUS	0.319	0.134	0.080
PEGASUS without weight decay	0.418	**0.201**	**0.174**
BART	0.393	0.177	0.131
BART without weight decay	**0.419**	0.198	0.168

Fig. 9. ROUGE and BLEU scores for test dataset.

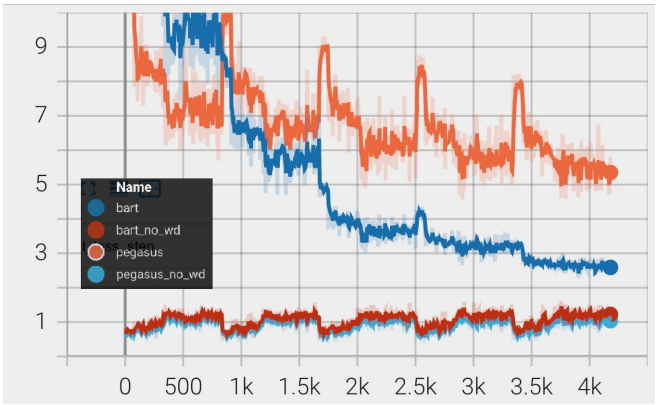

Fig. 10. Validation loss for BART and PEGASUS with and without weight decay.

	Test loss	
	Original	**Quantized**
PEGASUS	5.35	6.45
PEGASUS without weight decay	1.04	1.0
BART	2.66	12.59
BART without weight decay	1.22	10.07

Fig. 11. Test loss with and without quantization.

	Test loss				
	Prune rate 0%	**Prune rate 20%**	**Prune rate 40%**	**Prune rate 60%**	**Prune rate 80%**
PEGASUS	5.35	14.28	**12.93**	13.35	15.07
PEGASUS without weight decay	1.04	13.83	**13.23**	18.84	21.92
BART	2.66	14.47	**13.37**	14.43	13.56
BART without weight decay	1.22	11.41	10.98	11.31	**10.82**

Fig. 12. Test loss with different pruning ratios.

Model	ROUGE-1	ROUGE-2	BLEU	AVG Inference Time	Model Size
Pegasus-large fine-tuned on cnn_dailymail and quantized	0.360	0.1533	0.1134	4.58s	0.9281 GB
Pegasus-large fine-tuned on cnn_dailymail	0.418	0.2	0.174	9.2s	6.4 GB

Fig. 13. Comparison between quantized and non-quantized model.

special tokens. However, adding new special tokens altered the vocabulary and embedding size. We attempted to integrate these, but due to limited computational resources, we were compelled to focus our efforts on other aspects. The vocabulary size for pre-trained models is fixed, making training from scratch an infeasible option. Resource limitations further compounded our challenges. Our models were enormous, encompassing approximately 500 million parameters, and we dealt with a large training dataset of around 287,000 news articles and summaries. Distributed training was restricted to just 4 GPUs due to Google Cloud's quota limit, and these resources were largely unavailable, though slightly easier to access over the weekends. Lastly, in terms of performance evaluation, although ROUGE and BLEU scores are industry standards, we found that they may not fully encapsulate the human perception of the quality or coherence of the generated summaries, presenting yet another challenge in our work.

IX. Open Source Contribution

In the spirit of open science and to ensure complete transparency and reproducibility of our research, we have made all relevant code and models publicly available. The entire suite of code developed for this study can be accessed at our GitHub repository **https://github.com/mahalrs/newsum**. Further, we have also utilized the Hugging Face platform to share our trained models. For those interested in performing inference with our models without going through the hassle of setting up a local environment, we have created a Hugging Face space. This will allow for an easy and direct use of our models online. You can access this at **https://huggingface.co/spaces/aloutzidis/newsum** and the model id can be accessed via the Hugging Face handle **aloutzidis/pegasus-large-cnn-quantized**. These resources are intended to foster further research and development in the community, and we welcome any feedback or contributions.

X. Conclusion

This paper has highlighted our comprehensive approach to fine-tuning a Large Language Model on a news dataset. We successfully performed distributed training across four GPUs, demonstrating the feasibility and efficiency of this technique in dealing with hundreds of millions of parameters. We further optimized the PEGASUS model through the implementation of quantization and pruning, thereby enhancing the model's performance while reducing its computational requirements. We also performed benchmarking on PEGASUS and quantized PEGASUS to offer a comparative understanding of the performance of these models. However, our initial plan was to incorporate additional features such as named entities, topic modeling, and sentiment analysis as additional input to the models to improve the quality of generated summaries. As we started working on the project, we performed named entity recognition and topic modeling as a data preprocessing step. We planned to input these additional features using special tokens, however, we didn't realize in the beginning that adding additional special tokens will require us to change the size of the embedding layer as it changes the vocab size. As a result, incorporating additional features such as named entities was deemed unfeasible as it would have required training from scratch, which was beyond our resource limitations. Our best-performing model was the pegasus-large without weight decay with quantization, illustrating the effectiveness of this configuration. Lastly, we are proud to have contributed back to the open-source community, facilitating further advancements in the field. Our work, though challenging, has established a robust foundation for future research into the application and optimization of Large Language Models for news summarization tasks.

References

[1] T. B. Brown et al., "Language Models are Few-Shot Learners," arXiv.org, May 28, 2020. https://arxiv.org/abs/2005.14165v4 (accessed May 11, 2023).

[2] R. Nallapati, B. Zhou, C. N. dos Santos, C. Gulcehre, and B. Xiang, "Abstractive Text Summarization Using Sequence-to-Sequence RNNs and Beyond," arXiv.org, Feb. 19, 2016. https://arxiv.org/abs/1602.06023v5 (accessed May 11, 2023).

[3] M. Lewis et al., "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension." arXiv, Oct. 29, 2019. doi: 10.48550/arXiv.1910.13461.

[4] A. Vaswani et al., "Attention Is All You Need." arXiv, Dec. 05, 2017. doi: 10.48550/arXiv.1706.03762.

[5] J. Zhang, Y. Zhao, M. Saleh, and P. J. Liu, "PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization." arXiv, Jul. 10, 2020. doi: 10.48550/arXiv.1912.08777.

[6] Lin, Chin-Yew. (2004). ROUGE: A Package for Automatic Evaluation of summaries. Proceedings of the ACL Workshop: Text Summarization Braches Out 2004. 10.

[7] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a Method for Automatic Evaluation of Machine Translation," in Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, Jul. 2002, pp. 311–318. doi: 10.3115/1073083.1073135.

[8] L. Biewald, 'Experiment Tracking with Weights and Biases'. 2020.