Journal to Wiki Text Style transfer: Simplifying the medical literature for broader comprehension

Team Word Nerds

Alex Jonas, Annie Lam, Matthew L. Senjem, & Luis Silva
University of Minnesota
Minneapolis, MN 55455, USA
jonas060@umn.edu,lam00058@umn.edu,senje001@umn.edu,silva364@umn.edu

1 Introduction / Motivation

The motivation for translating medical literature to a Wikipedia structure is to simplify the medical literature. Translating a medical research paper to a Wikipedia article is akin to summarizing the paper's major findings in a digestible format. The simplification and increased digestibility of a Wikipedia article would allow broader comprehension for individuals not privy to reading medical research and to healthcare providers who need to absorb a large amount of journals in a limited amount of time. Thus, the laity and scientific researcher alike are given another tool in their personal arsenal for interpreting and making sound decisions based on the latest medical research in a timely manner.

1.1 Proposed Idea and Hypothesis

1.1.1 Problem definition

The problem we are aiming to solve is to collect a dataset of scientific papers and corresponding Wikipedia articles and use this dataset to build an automated system for generating Wikipedia-style articles from scientific papers. We will collect a dataset of scientific papers, with corresponding wiki or blog posts summarizing the articles in wiki form. We will then use the resulting dataset to train a model for performing the wiki-style summarization task for each article. We will explore various options for the model architecture, e.g. fine tune a pre-trained model from one of the references mentioned in the literature survey section above.

1.1.2 Initial Idea

Our main idea is to create a NLP tool to summarize medical research papers and transform their style into a wiki format. (Chen et al., 2020) The idea is that by building a model based on a parallel dataset of medical articles and wiki summaries scraped from the web, we will be able to simplify texts from the medical literature for a wider audience.

1.1.3 Hypothesis

Our working hypothesis is that building an NLP model can effectively summarize and transfer styles from medical literature to wiki texts is possible. (Toshevska and Gievska, 2022) We believe that an existing NLP model, perhaps with pre-training on an existing similar dataset and task, can be leveraged to perform this task and fine-tuned using the dataset we will collect from parallel examples of previous papers summarized as wiki texts.

1.2 Broader Impact

The impact of simplifying medical literature would be increased scientific literacy and engagement with science and technology. This could be seen in both healthcare providers and the lay population.

Simplifying the medical literature for healthcare professionals will profoundly impact the effectiveness of their care. Physicians and other professionals particularly those in training, often require a generalized understanding of a paper. By having multiple "wikified" articles and NLP tools that can "wikify" any article a provider needs - the literature review required for clinical practice can be made more efficient with the results from our project.

Regarding the lay population, having articles in the style of Wiki pages can increase the readability of medical literature. The argument here is that people in general are more familiar with wiki pages than journal articles. As a result, transferring papers to the wiki-style might increase the readability and interpretability of studies for non-medically trained individuals interested in the published literature.

2 Methods/Approach

2.1 Data Collection

We explored a few different options for collecting our data set. One option was using R Selenium, and another option, suggested by our mentor Zae, was using scrapy in combination with xpaths to get the summary and the link to a corresponding PDF file. The set of PDFs be interpreted using PyPDF, resulting in the PDF along with the corresponding wiki text being split into training and test sets. Initially we would access the Random page link from the Wiki Journal Club, extract the text on the wiki page as our "Wiki text" - label, and extract the text from the linked PDF as "Original Text" - input.

The process we decided on was Beautiful Soup to initially scrape all links from the web page containing a list of all usable articles. Beautiful Soup scraping filled a data frame with article links and acronyms. We then used the same package to access each link and add the wiki text our data set as our label. The links for the PDF and the full article were also scraped and organized as "inputs" for our data set.

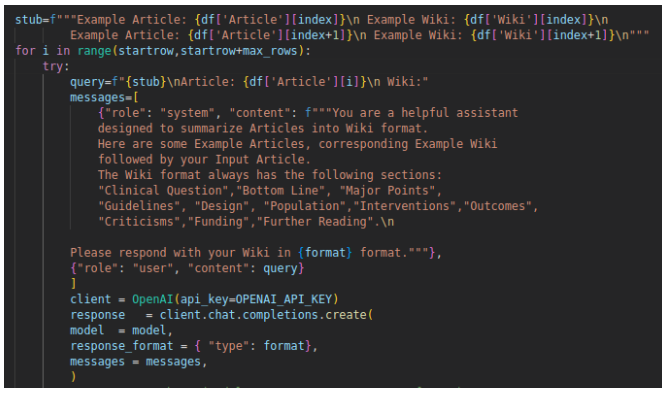
The initial idea was to utilize the PDF file original articles as our data set input. The issue is with utilizing the PDF files is it would require all PDF files to be saved in a local machine and subsequently extract the text into the data set in the correct rows. After a few attempts, it was decided that this approach would be unfeasible in the time frame required for the project to be completed.

The alternative was to use R Selenium to scrape the original text from the article's web page and iteratively place the scraped text into the data set. All articles published by the New England Journal of Medicine were collected with R Selenium. The task required multiple code runs. R Selenium runs through GitHub, and there is a limit to the number of virtual machines a free account can open in two hours. Once the limit is reached, the R function halts, and two hours of wait time begin before the code can be run again. Given the GitHub limitations, it took nearly two days to scrape 356 articles from the New England Journal.

The R Selenium approach possesses a few fail cases including limited scalability. R Selenium is webpage-specific if an article from other journal websites were to be extracted, a specific xpath would need to be written for each. Limiting our current configuration's usefulness to other journals and publications. All scraped articles were free, but most current published literature is protected by copywrites and paywalls. Selenium scripts that log into accounts permitted to access these articles would have to be written, thus limiting the usefulness of the current configuration in scaling data collection to a broader scope. The lack of scalability given our current R Selenium configuration is troublesome when considering the reproducibility of our current R Selenium configuration. Particularly if other researchers don't have access to journals protected by paywalls.

2.2 Few-shot Prompting With GPT-4

After some initial experiments with using various models to summarize articles, we quickly realized that the length of the input articles, up to 47729 characters, was way too large for most of the available models. Acting on a suggestion from our mentor, Zae Myung Kim, we decided to try prompting with GPT-4-1106-preview, using the OpenAI API. After some experimentation, we developed a two-shot prompt that produced remarkably accurate outputs in the desired format. Specifically, we designed a prompt that first showed two examples of input articles and their corresponding wiki pages to GPT-4, and then gave a third article and asked GPT-4 to produce a wiki page like the two examples given.



```
stub=f"""Example Article: {df['Article'][index]}\n Example Wiki: {df['Wiki'][index]}\n
        Example Article: {df['Article'][index+1]}\n Example Wiki: {df['Wiki'][index+1]}\n"""
for i in range(startrow,startrow+max_rows):
    try:
        query=f"{stub}\nArticle: {df['Article'][i]}\n Wiki:"
        messages=[
            {"role": "system", "content": f"""You are a helpful assistant
            designed to summarize Articles into Wiki format.
            Here are some Example Articles, corresponding Example Wiki
            followed by your Input Article.
            The Wiki format always has the following sections:
            "Clinical Question","Bottom Line", "Major Points",
            "Guidelines", "Design", "Population","Interventions","Outcomes",
            "Criticisms","Funding","Further Reading".\n

        Please respond with your Wiki in {format} format."""},
        {"role": "user", "content": query}
        ]
        client = OpenAI(api_key=OPENAI_API_KEY)
        response   = client.chat.completions.create(
        model  = model,
        response_format = { "type": format},
        messages = messages,
        )
```

Figure 1: 2-shot prompt used with GPT-4

In addition to providing two examples, we instructed GPT-4 that the "wiki format always has the following sections: "Clinical Question", "Bottom Line", "Major Points", "Guidelines", "Design", "Population", "Interventions", "Outcomes", "Criticisms", "Funding", "Further Reading", as shown in Figure 1. We believe, and observed empirically, that calling out the section titles in this way helped the GPT-4 model to identify which parts of the text to attend to in figuring out the mapping from article to wiki using the provided examples. (Mitchell et al., 2023) (Liu et al., 2023) After developing the 2-shot prompt, we wrote Python code and notebooks to iterate through the list of collected articles and generate wiki text for each one. Next, we wrote these responses to individual text files and generated a webpage of the results, which can be viewed at: msenjem.github.io/Journal2Wiki. The initial version of the webpage was generated by a GPT-4 assistant in the OpenAI playground and can be viewed at: msenjem.github.io/Journal2Wiki/index_gpt.html. All of the code for the 2-shot prompting, website generation, and other aspects of the project can be found at github.com/msenjem/Journal2Wiki

2.3 Encoder-Encoder-Decoder Model

2.3.1 Overview

We sought to create output similar to ChatGPT-4 with prompt engineering by fine-tuning an architecture of our own. The architecture utilized was an "Encoder-Encoder-Decoder" model. Only our encoder-decoder model would be fine-tuned, while our initial encoder model would remain unchanged.

2.3.2 Data set

The data set for the architecture utilized our newly created Research Paper data set with a slight twist. The twist is that each research paper was broken down into its "components," i.e., "Results," "Methods," "Discussion," and "Abstract." We removed the "Abstraction" component from our data set as our team concluded the abstract already summarized the entirety of the research paper, which we deemed unjustly aiding the model. The "label" or intended result for the model would remain the same, that being the wiki-journal layout of the research paper.

2.4 Architecture

The Encoder-Encoder-Decoder Model architecture consisted of an initial encoder model. Our team decided on the "Bert-Based-Uncased" model (Devlin et al., 2019). The result from the Bert model would be fed into the Encoder-Decoder model. The rationale behind the use of the Bert model was the encoding capacity of Bert along with its specification for the classification of text. Our team believed Bert's classification potential would be useful in determining the characteristics unique to each component of the research paper and classifying contextually important texts (Tran et al., 2022). Given an input component, the result of our Bert Model was 12 hidden states corresponding to Bert's 12 hidden layers. The 12 hidden states were then concatenated, and token vectors were retrieved, giving us a matrix of words by vector embedding of each word (w x v). We then take the mean value for each vector value, resulting in a (1 x v) embedding vector representation of our component. We follow the same formatting for each element with a final representation of our research paper as three components by embedding vectors for each component (3 x v).

The three component embedding vectors were concatenated, tokenized, and fed into our encoder-decoder model. Our encoder-decoder model of choice was Pegasus-xsum (Zhang et al., 2020) (Phang et al., 2022) given its Long String summarizing capability with a 16,000 token input limit. The result, given a cross-entropy loss function and Lrouge scoring, would eventually be the summarization of the research papers' "Results," "Methods," and "Discussion" components following a formatting similar to that of our wiki journal label.

3 Results

3.1 ROUGE Score

For automatic evaluation of the GPT-4 generated wiki pages, the ROUGE Score was calculated using HuggingFace's Evaluate library, which uses Google Research's re-implementation of ROUGE (noa). Also known as the Recall-Oriented Understudy for Gisting Evaluation, ROUGE was first introduced in 2003 as an automatic evaluation metric for summarization (Lin and Hovy, 2003). The ROUGE metric evaluates the n-gram overlap between the machine-generated text and the human-generated text.

rouge1	rouge2	rougeL	rougeLsum
35.37	14.28	17.69	23.23

Table 1: ROUGE scores calculated on GPT-4-generated vs. human-generated wiki pages. rouge1:unigram scoring, rouge2: bigram scoring, rougeL: longest common sub-sequence scoring, and rougeLSum: uses text split on "$\backslash n$"

3.2 Human Evaluation

While the ROUGE score provides a general overview of the similarity between the human and machine-generated texts, it cannot adequately capture all summarization quality metrics. The addition of human evaluation provides additional insight to create a more comprehensive evaluation of the overall quality and usability of the generated text. The human evaluation criteria were measured using the Likert scale. Group members evaluated the following human evaluation metrics for 5 summaries: fluency, clarity, accuracy, and comprehensiveness. Fluency and clarity were measured for the text as a whole, and accuracy and comprehensiveness scores were evaluated independently for the major sections.

Fluency refers to the quality of the sentence structure, word choice, and grammar, and clarity is the measure of ease of interpretability. Accuracy is the measure of whether the information included is true, and comprehensiveness is a measure of how well the generated text captures the breadth of information included in the human-generated text.

Human Evaluation	*Likert Score*
Overall	
Fluency	4.25
Clarity	4.15
Clinical Question	
Accuracy	4.7
Comprehensiveness	4.7
Bottom Line	
Accuracy	4.65
Comprehensiveness	4.55
Major Points	
Accuracy	3.65
Comprehensiveness	3.25
Guidelines	
Accuracy	3.2
Comprehensiveness	3.05
Design	
Accuracy	4.5
Comprehensiveness	4.5
Population	
Accuracy	3.85
Comprehensiveness	3.2
Interventions	
Accuracy	3.7
Comprehensiveness	3.2
Outcomes	
Accuracy	4.3
Comprehensiveness	3.35
Criticism	
Accuracy	2.5
Comprehensiveness	2.4

The overall fluency and clarity of the generated text scored well. However, it should be noted that the sentences were more convoluted than the human-generated texts. The accuracy and comprehensiveness scored better for shorter sections such as the clinical question and bottom line. Scores were lower for longer, more detail-oriented sections such as the major points and outcomes. As a whole, the generated text was successful at summarizing the major takeaways of the research article, but often lacked the specificity of the human-generated texts. For example, the wiki journal posts included more numerical results and additional context. The generated articles can be viewed at the results website.

3.3 Failure Cases

The failure cases for this project included some incorrect details in the generated summaries, occasional wrong numerical data, and lack of depth in the "Major Points" sections. On a couple of occasions, the model generated empty sections in the wiki, or extremely sparse one-line summaries. The model also failed to generate all the major categories in some instances, so some articles were missing sections. This issue could likely be addressed by providing more training examples and increasing prompt specificity. The prompts used included multiple instructions in a single prompt, which can cause the model to miss steps. We hypothesize that the model may produce more precise results if the instructions were given one at a time, and plan to test that approach in future work.

4 Discussion

Regarding our Encoder-Encoder-Decoder architecture, we ran into a few hurdles. Although Pegasus-xsum allowed for large tokenized inputs - up to 16,000 - the Bert model allowed only 4,000. Thus we were limited in our ability to input a research component into the initial Encoder Bert model. To solve we split our component into sentences. The sentences were then tokenized, fed into our Bert Model for our embedding vector output then summed by all embedding vector sentences in the component. The result of summing embedding vector sentence text is a substantial loss in context and thus a potentially worse input to our encoder-decoder Pegasus-xsum model than simply tokenizing the component and inputting directly to Pegasus-xsum.

The other problem our team faced which we were unable to solve was the Pegasus-xsum decoder outputting numerical values rather than token predictions. The problem we believe stems from how we created our embedding vector representations. Summing the sentence embedding vector of each sentence within our component resulted in a loss of word contextualization. It could be argued we lost nearly all the component word contextualization and thus are left with garbage input component embedding vectors into our encoder-decoder Pegasus-xsum model. Resulting in useless output numerical values rather than tokenized predictions.

5 Ethical considerations

There are relevant ethical considerations to be brought up with this work. Healthcare providers may use the model's generated texts as a reference for clinical practice, but inconsistent output may lead to erroneous medical decisions. As such, it would be important to make sure the users of the tool are informed about this possibility through disclaimers. These disclaimers should only be removed if a separate study evaluating the quality of the scientific evidence produced by the model is tested and peer-reviewed.

Another relevant consideration is copyright. Summarizing articles that are copyright protected and distributing them for free might be seen by journal publishers as an infringement of authorship law. As such, one could consider summarizing only articles that are not protected by paywalls, as this current analysis has done.

6 Future Work

For future work on the GPT-4 prompting method, we would like to do some further refinement and variations of our prompting method and compare the results between different prompt setups. For example, we could compare the results from our current 2-shot approach to a 3-shot approach, as well as a 1-shot approach, and we could explore breaking up the prompt into multiple sub-prompts, creating separate prompts for the sections, "Clinical Question", "Bottom Line", etc. In addition, we plan to try similar methods using newer models as they arise, e.g. GPT-5 and other larger and larger models that will no doubt inevitably become available.

For future work on the Encoder-Encoder-Decoder method, we would like to include our initial encoder BERT in fine-tuning our Encoder-Encoder-Decoder model. BERT fine-tuning would be straightforward as the self-attention mechanism in the Transformer allows Bert to model many downstream tasks. (Devlin et al., 2019)

References

Rouge - a hugging face space by evaluate-metric.

Mingda Chen, Sam Wiseman, and Kevin Gimpel. 2020. Generating wikipedia article sections from diverse data sources. *CoRR*, abs/2012.14919.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Chin-Yew Lin and Eduard Hovy. 2003. Automatic evaluation of summaries using n-gram co-occurrence statistics. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 150–157.

Chao Liu, Xuanlin Bao, Hongyu Zhang, Neng Zhang, Haibo Hu, Xiaohong Zhang, and Meng Yan. 2023. Improving chatgpt prompt for code generation.

Melanie Mitchell, Alessandro B. Palmarini, and Arseny Moskvichev. 2023. Comparing humans, gpt-4, and gpt-4v on abstraction and reasoning tasks.

Jason Phang, Yao Zhao, and Peter J. Liu. 2022. Investigating efficiently extending transformers for long input summarization.

Martina Toshevska and Sonja Gievska. 2022. A review of text style transfer using deep learning. *IEEE Transactions on Artificial Intelligence*, 3(5):669–684.

Loc Hoang Tran, Tuan Tran, and An Mai. 2022. Text classification problems via bert embedding method and graph convolutional neural network.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. 2020. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization.