UMIACS Wiki Chatbot

Prepared for
CMSC 473/673

Prepared by
Andrew Wang, Dongyang Zhen, Rodrigo Sandon,
Sahil Gaba, Sanjali Yadav, Tanvi Simhadri

University of Maryland, College Park

Date: Dec 13, 2024

Contents

1. Motivation

The UMIACS wiki contains a wealth of valuable information, but its organization across numerous pages can make finding specific details challenging and time-consuming for users. Developing a chatbot to address this issue presents a transformative solution that not only enhances accessibility but also optimizes user efficiency. By enabling users to ask questions in natural language and receive precise, context-aware responses, the chatbot eliminates the frustration of sifting through multiple pages manually. For questions that are beyond the scope of the wiki and require human expertise, the chatbot directs users to contact UMIACS staff, ensuring that complex or unique issues are addressed adequately. Thus, this user-friendly tool would save time, reduce barriers to information, and make the wiki's resources more approachable. It will also alleviate the burden on UMIACS staff by automating responses to repetitive inquiries, enabling them to concentrate on more complex tasks. Ultimately, this innovation would amplify the value of the UMIACS wiki, creating a more dynamic and efficient resource for the users.

2. Literature Review

Chatbot development has advanced significantly with recent breakthroughs in natural language processing research. Several state-of-the-art techniques are now available for building highly capable chatbots. While fine-tuning pre-trained language models and Retrieval-Augmented Generation (RAG) are among the most prominent and effective methods for chatbot development, other innovative approaches, such as transfer learning, domain adaptation and zero-shot learning are also being used to create versatile and domain-specific conversational agents. For our chatbot development, we look at two main approaches and conduct a literature review.

2.1 Fine-tuning on Pre-trained Models

Vulić et al (2021) propose a two-stage fine-tuning approach to enhance pretrained models for chatbot applications. First, the model is fine-tuned on general conversational datasets to adapt to dialogue-specific dynamics, followed by task-specific fine-tuning to meet application requirements. Data augmentation techniques, such as synthetic dialogue generation and paraphrasing, improve training data diversity and robustness. Evaluations show ConvFiT significantly enhances response fluency, coherence, and task success rates, outperforming baseline methods. This framework effectively adapts pretrained models for scalable and versatile chatbot deployment.

There are comprehensive frameworks like LlamaFactory that support efficiently fine-tuning and customizing large language models (LLMs) like GPT, LLaMA, and OPT for specific tasks and domains. LlamaFactory supports over 100 models and leverages parameter-efficient techniques such as LoRA and adapter layers to reduce computational costs. Featuring a user-friendly interface, LlamaBoard, it streamlines fine-tuning, model selection, and benchmarking while supporting data augmentation for robust training. With tools for scalable deployment and multi-model experimentation, LlamaFactory simplifies the development of chabots.

2.2 Retrieval-Augmented Generation (RAG)

Lewis et al. (2020) introduced RAG as a method to overcome the limitations of traditional pre-trained language models, which, despite encoding factual knowledge, often struggle with efficient knowledge retrieval and manipulation. RAG integrates parametric memory (via a pre-trained seq2seq model) with non-parametric memory (using a dense vector index like Wikipedia), facilitated by a neural retriever. This dual-memory approach allows RAG to access and incorporate external knowledge explicitly, enhancing the model's performance in knowledge-intensive tasks such as open-domain question answering (QA). The paper distinguishes between two RAG methodologies: one that uses a consistent set of retrieved passages across a generated sequence and another that retrieves different passages for each token. Their findings indicate that the latter approach, which dynamically retrieves context-specific passages, significantly improves performance over static methods, demonstrating RAG’s effectiveness in generating coherent and relevant responses.

Ke et al. (2024) focus on the practical implementation and evaluation of RAG in various settings, showcasing its adaptability and performance improvements. Although their case study is set in a medical context, the methodologies and findings are broadly applicable. They demonstrated that an RAG-enhanced GPT-4.0 model could achieve a significant accuracy improvement over standard models, with results showing a substantial increase in performance when RAG was applied. The case study emphasizes the efficiency of RAG models, noting that they can generate answers more quickly compared to traditional methods. This efficiency, coupled with improved accuracy, underscores the potential of RAG for enhancing LLMs' performance across different applications, reinforcing the benefits of incorporating external knowledge sources.

There are open-sourced frameworks like RAGFlow designed to facilitate Retrieval-Augmented Generation (RAG) by integrating large language models (LLMs) with deep document understanding. It streamlines the RAG workflow, enabling businesses to extract accurate, citation-backed information from complex, unstructured data sources such as documents, images, and web pages. One of the standout features of RAGFlow is its template-based chunking, which makes the document segmentation process more explainable and flexible, allowing for human intervention if needed. This ensures that the generated responses are not only

accurate but also grounded in verifiable sources, reducing hallucinations often seen in LLMs. Moreover, its compatibility with heterogeneous data sources, including text, images, and structured data, makes it a versatile solution for businesses seeking to integrate RAG workflows into various applications.

2.3 Hybrid Approaches

Rangan et al. (2024) combine the capabilities of retrieval-augmented generation (RAG) with the precision of a vector database and the nuanced understanding of a fine-tuned large language model (LLM). The process starts by extracting text from a PDF document using PyMuPDF, which is then processed in two parallel pathways: one creates a vector database with Chroma for efficient content retrieval, and the other prepares the data for fine-tuning the LLM on a customized dataset. This dual-pathway approach integrates structured data retrieval with the contextual learning of the LLM.

When a user submits a query, the algorithm engages both pathways to generate responses. The vector database retrieves relevant content based on similarity, producing a direct, data-driven answer ("answer 1"), while the fine-tuned LLM generates a context-aware response ("answer 2"). These two answers are synthesized to leverage the precise retrieval from the database and the fine-tuned LLM's nuanced understanding. This combined content serves as input to a foundational LLM, which, although not fine-tuned for the task, uses the synthesized insights to generate a final response that is both accurate and contextually rich.

Kulkarni et al. (2024) presents a RAG-based approach for building a chatbot that leverages FAQ data to answer user queries, optimized for cost efficiency and retrieval accuracy. The system integrates an in-house retrieval embedding model trained using infoNCE loss, which outperforms general-purpose public models in both retrieval accuracy and out-of-domain (OOD) query detection. The chatbot uses the GPT-35-Turbo model as its LLM for generating responses, while retrieval optimization is achieved through a reinforcement learning (RL)-based policy model external to the RAG pipeline. This model dynamically decides whether to fetch FAQ context ([FETCH]) or skip retrieval ([NO_FETCH]) based on the query and prior context, reducing token usage and associated costs.

The RL model, trained using a policy gradient approach, evaluates its actions through GPT-4, which serves as a reward model by rating the quality of chatbot responses. Reward shaping ensures that accurate, cost-efficient answers receive positive rewards, while poor decisions (e.g., skipping retrieval when necessary) are penalized. Experiments with GPT-2 and an in-house BERT model as the policy backbone achieved significant (~31%) cost savings while slightly improving accuracy. Although demonstrated on an FAQ chatbot, this generic RL-based optimization approach can be adapted for any existing RAG pipeline.

3. Project Goals

The primary goal of this project was to develop two chatbot systems capable of answering questions based on information from the UMIACS wiki. To achieve this, we set out to:
Build tools to scrape and extract data from the UMIACS wiki.
Process the extracted data into structured formats suitable for training and evaluation.
Implement systems that can effectively answer questions using both retrieval-based and model-based methods.
Establish a comprehensive benchmark for evaluating the performance of question-answering systems on wiki-based datasets.
These goals aimed to create robust and scalable solutions for wiki-based information retrieval and question answering.

4. Approach and Results

Our approach involves exploring two primary techniques—Retrieval-Augmented Generation (RAG) and fine-tuning—for developing the UMIACS Wiki chatbot. We detail the iterative process undertaken to refine and optimize each of these models, highlighting the challenges and improvements at each stage. Finally, we introduce a hybrid approach that integrates RAG with fine-tuning.

4.1 Scraping the Wiki

To accurately and efficiently scrape the UMIACS wiki, we implemented a Python-based scraping pipeline, utilizing libraries such as BeautifulSoup to navigate the wiki's structure and extract relevant information. The scraper was originally designed to recursively traverse the wiki, identifying and capturing content from linked pages while preserving the hierarchical organization of the data; however, this wiki had a page containing links to all other pages. We used that page to scrape all pages on the wiki and tossed out any redirects that lead to the same page. Special attention was given to parsing structured elements such as tables, headings, and lists to maintain the semantic relationships present in the wiki. We also ignored images present on the wiki as they usually contained information that was already present in the wiki’s text. The extracted content was preprocessed to remove irrelevant data, standardize formatting, and prepare it for downstream tasks, such as question-answering and training models. Error-handling mechanisms were integrated to address potential issues like missing pages or inconsistent formatting, ensuring a robust and reliable scraping process.

4.2 Retrieval-Augmented Generation (RAG)

The process of implementing retrieval augmented generation involves converting the document texts into vectors, then storing the vectors with the original text in a database for future querying. When the chatbot receives a question, it translates this query into a vector, then performs a similarity search on the vectors that were previously stored in the database. The original text from the retrieved vectors are then provided to the LLM as context, which improves the accuracy of the output. There are multiple libraries that can be used for RAG, and we decided to use LangChain due to its integrations with different vector databases and support for LLM pipelines.

To begin with, we manually split our documents into 128 token chunks, and converted them into vectors using the sentence-transformers/all-MiniLM-L6-v2 embedding model, which has a maximum context length of 256 tokens and vector embeddings with 384 dimensions. The reason we use 128 tokens is because the model was trained using sequences of 128 tokens or shorter. The two main drawbacks of this approach was that sentences were typically split between chunks, and 128 would usually not include enough context.

In order to rectify these issues, we switched to using LangChain's RecursiveCharacterTextSplitter, which attempts to keep size within a target size by only splitting on specific separators. By default, it prefers to separate on "\n\n", "\n", " ", and "", in decreasing order of priority. However, we noticed that the chunks it produced were sometimes far greater than 128 tokens. This was primarily because when we converted the original HTML into raw text, we did not add in newlines. Since HTML elements themselves are what creates line breaks, we just had to add newline characters whenever those breaking elements were seen. Additionally, we combine consecutive newlines into one or two newlines. We tried different numbers of consecutive newlines to convert, and the line breaks in our translation are close to how sections look in the browser. From our testing, this kept paragraphs and short sections within single chunks.

Even with a more advanced chunking method, some query results were suboptimal. One in particular is the query, "how do I run a python notebook in nexus?". Parts of the true answer do not include words related to python notebook, so those chunks are not included. The model also seemed to consider "python notebook" and "jupyter notebook" as not very similar. Furthermore, chunks that mention "Nexus" a lot would be considered equally similar, and those would be included, even though they are not relevant to the question. We were using a very small model with very clear limitations, particularly the 128 token context length, so we found a different model on Hugging Face's Massive Text Embedding Benchmark. We selected dunzhang/stella_en_1.5B_v5, as it was the highest performing model that could fit on a single GPU in the Nexus class account partition. Although this model's maximum context size is 8192 tokens, its training samples were limited to 512 tokens, so this is our effective maximum chunk

size. In addition, the model supports separate sentence-to-sentence (s2s) and sentence-to-paragraph (s2p) embedding modes. In this context, s2s is converting paragraphs to vectors that represent their meaning. On the other hand, s2p converts questions to embeddings that are similar to answers. This is important as questions just do not have the same meaning as paragraphs, so a different translation is required.

Since we were now using a far larger model, we had to use Nexus to do anything regarding our RAG solution. Initially, we were using Redis as our vector database, which required a separate instance to be running. In order to be able run on Nexus, we changed our implementation to use Qdrant instead, as it runs within a python script.

This upgraded RAG system performed better in general, as larger chunks meant a lower chance of missing relevant chunks nearby. However, it still did not perform well on the example query above. Although the model itself was more accurate, which in this scenario is indicated by "python notebook" being considered more similar to "jupyter notebook," since the chunks are larger, chunks that mention "Nexus" even more are even closer. In general, with larger chunks, precise matches to questions are unlikely, as only a small portion of the chunk will be similar to a short question, and the rest of the chunk will change its meaning to be less specific.

As such, the final solution we came up with is to shorten the chunks to about a few sentences or ~100-300 tokens to maintain the precise meaning of shorter chunks. Each chunk then references the full page it came from, and ultimately this is provided as the context. This works for our use case specifically, as individual questions can almost always be answered by a single page, and the large page is less than 10,000 tokens, which fits in the context window of modern LLMs. This gets the best of short chunks and long contexts in our environment.

Finally, the quality of the chat model's output depends on three main factors: the context provided, the prompt, and the model itself. The aspect that is the hardest to change and what our work has focused on is the context, and thus we will only examine the retrieval results for this section. It is far safer to include a bit of extra context instead of potentially missing out on some, so we chose to select the top 4 chunks. Since we are retrieving entire pages, there is a chance that some or all of the chunks come from the same page. This is ideal, as it means it is very likely that the answer is on that page.

Using the same query as before: "how do i run a python notebook in nexus?", the page it is from and distance scores of the 4 chunks are shown below, first using the s2s and then using the s2p embedding mode.

```
s2s embedding:
[('SLURM.html', [0.39104783535, 0.403365552425, 0.406363248825, 0.437749147415])]
```

s2p embedding:
[('SLURM.html', [0.340046286583, 0.347794413567, 0.348588466644, 0.382137238979])]

We can see that in both modes, SLURM is the only page that comes up, and this is the page that tells us how to run Jupyter notebooks on Nexus. We can also see that the s2p embedding has lower distance to the retrieved chunks, which is what we expected.

4.3 Fine-Tuning

To fine-tune the model, we used the Llama Factory, an all-in-one platform designed to facilitate the efficient adaptation of LLM to specific tasks. Llama Factory integrates a suites of advanced training methods (e.g. Llama3, LLaVA, Gemma, etc), enable users to customize the training process without extensive coding modifications. In this project, we choose LLaMA3-8B-Instruct as the base model. The 8B model could offer robust performance for instruction-following (Qusetion-Answer in our dataset), with the computing resources limit given by the Nexus server. Besides, the smaller base model accelerates the fine-tuning duration and is easy to update.

The first step was to modify the scraped UMIACS dataset, where we converted the chunked UMIACS text into Q/A pairs. To deal with the large volume of text data, we leveraged OpenAI API and ChatGPT-4O Mini to automate the generation of questions and corresponding answers. Each page of the UMIACS wiki was fed into ChatGPT with a refined prompt that was used to generate corresponding Q/A pairs. The next step in the process was LoRA fine-tuning, which enabled the efficient update of approximately 12 million parameters in the LLaMA3-8B-Instruct base model. The LoRA fine-tuning significantly reduced the training time, 2.5 hours to fine-tune on Nexus A5000 GPU, where regular fine-tuning takes more than 4 hours (3 hours are the Nexus limit for a single user to run a job using A5000). After the fine-tuning, we deploy the project on the Weibu & Hugging Face interface. The sample results are shown in Figure 1.

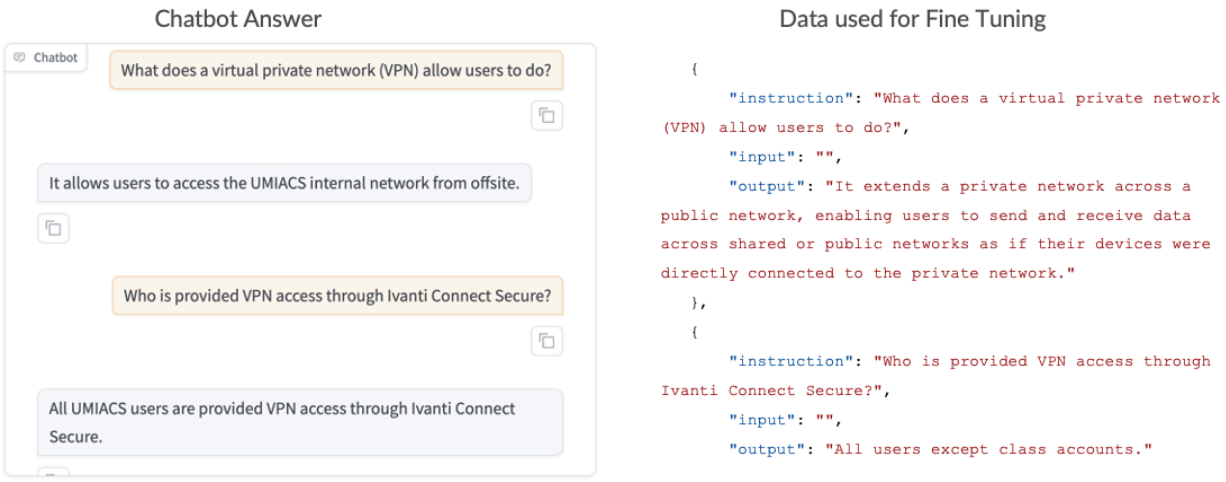

Figure 1. Fine-tuning chatbot results compared to training Q/A pairs.

After the initial round of fine-tuning, several problems were identified. First, the models' outputs were mostly brief and lacked the essential information for the comprehensive response. This issue is caused by the training dataset, as the Q/A pairs gendered for fine-tuning were relatively short, limiting the model's ability to produce elaborate solutions corresponding to UMIACS wiki related questions. Additionally, the fine-tuned model demonstrated a loss of contextual information, as it was trained separately on text, rather than the hierarchical structure in the original HTML file. So the background information is missing. Moreover, the image elements and information from external links are excluded. In summary, in our first version of fine-tuned chatbot, the output quality is considerably low. To overcome these limitations, we propose two pipelines illustrated in Figure 2.

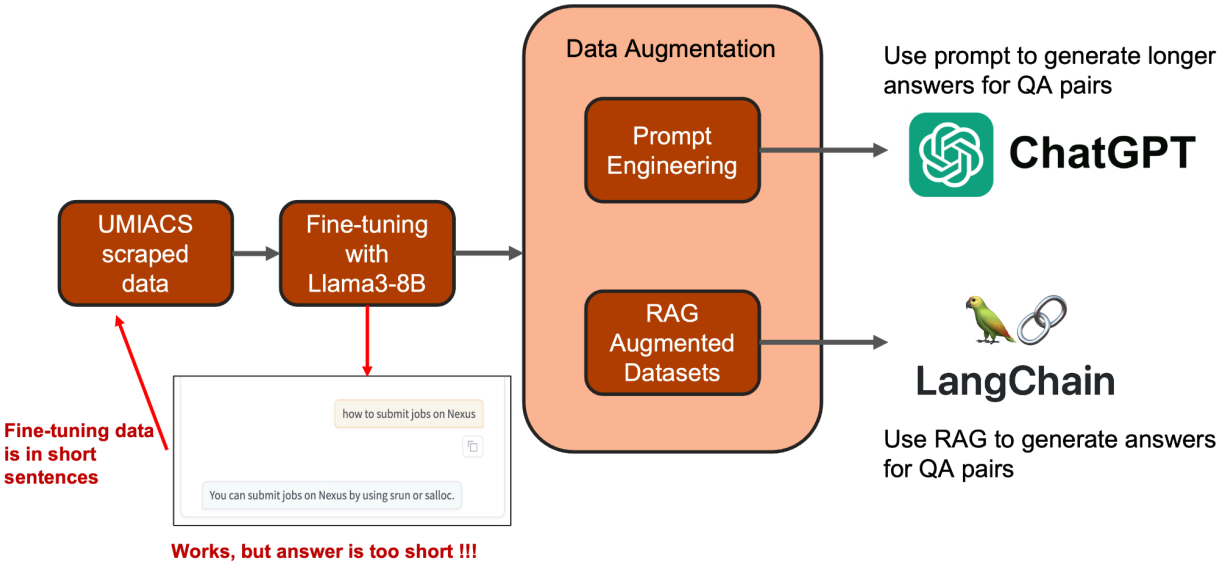

Figure 2. Fine-tuning improvement pipeline.

In the first direction, we adopted an RAG approach to augment the dataset to enhance the model's output performance. In detail, the Q/A pairs were reconstructed leveraging RAG, which can provide more detailed and contextually rich answers. By integrating information retrieved from the UMIACS wiki during the answer generation process, RAG ensured that the augmented dataset captured a broader range of details, including previously omitted contextual elements. The augmented dataset, now enriched with comprehensive and context-aware Q/A pairs, was used for a second round of fine-tuning. The comparison of the original answreing and the new answering are described in Figure 3. The model trained on RAG augmented dataset is much longer and elaborated than the original one.

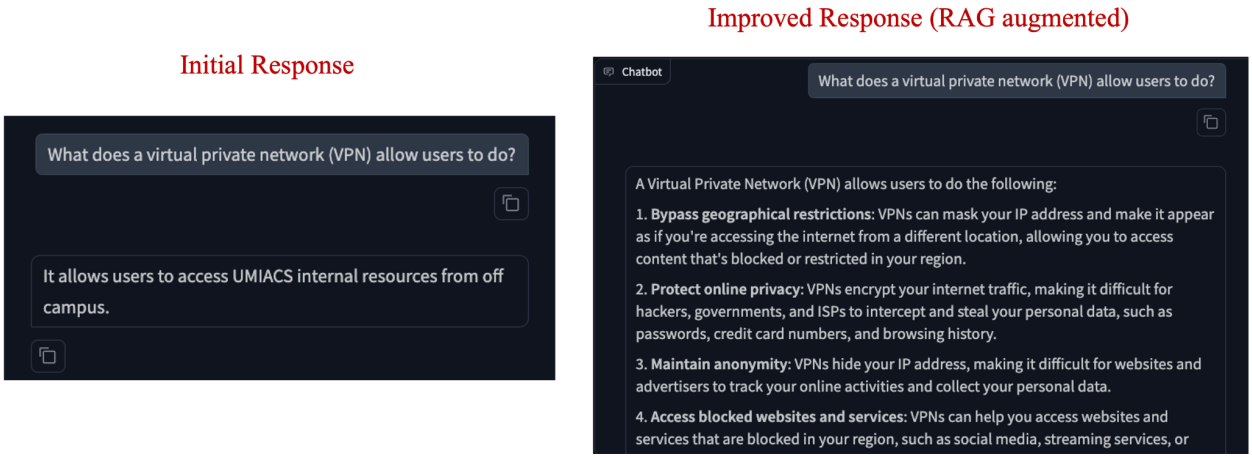

Figure 3. RAG augmented fine-tuning.

The second way to overcome the short-output problem is to use prompt engineering to refine the dataset by generating longer and more detailed answers. "Generate as many detailed, specific questions and answers as possible based on the provided text, ensuring each is unique and directly supported by the content."; "Expand your response by creating multiple detailed, context-focused questions and answers, targeting every distinct aspect of the text." such prompts were added to the original prompt when generating the Q/A pairs. Under this new dataset, our fine-tuned chatbot could able to generate longer content. However, missing the link and image information are still there. To solve this, we replace the image with the original website link, every time there are some images, we will redirect the user to the original page for details. The prompt engineering enhance output is shown below:

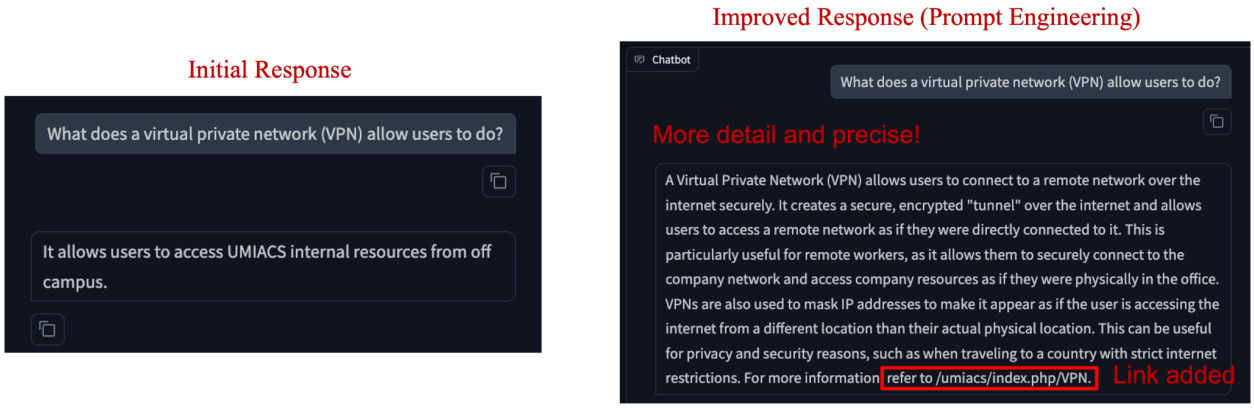

Figure 4. Prompt Engineering enhanced fine-tuning.

5. Evaluation

We conducted an evaluation to compare our chatbot performance against the state-of-the-art GPT4o model provided by OpenAI that operates without retrieval augmentation. The baseline operated solely on its training and the direct context we gave it, which was the intended document for the question we were asking it. This evaluation, then, essentially compares the current best solution out there, with a home-made solution using only open-sourced tools.

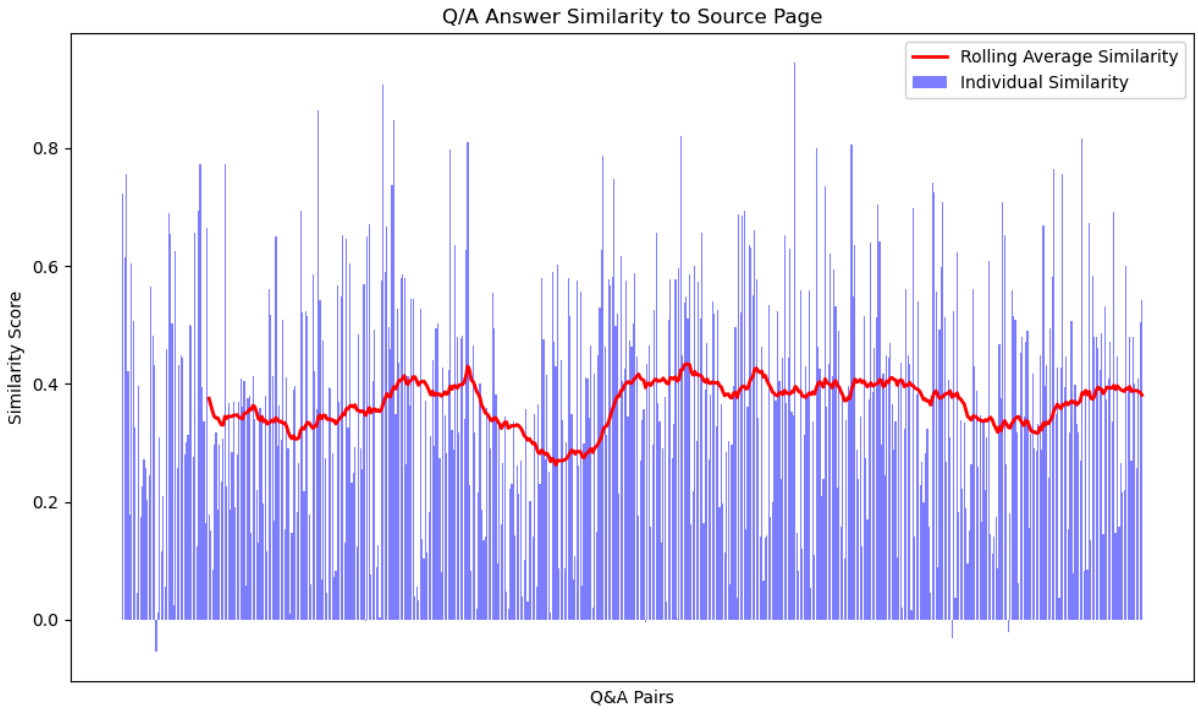

Figure 5: Cosine Similarity for our RAG Chatbot.

The figure displays cosine similarity of the answers given by the chatbot and its truth source (the document the question came from). The graph represents cosine similarity scores for over 700+ Q&A pairs. Individual score for each question in blue show considerable variability and the rolling average (red) of these scores seem to be stable around 0.3 to 0.45

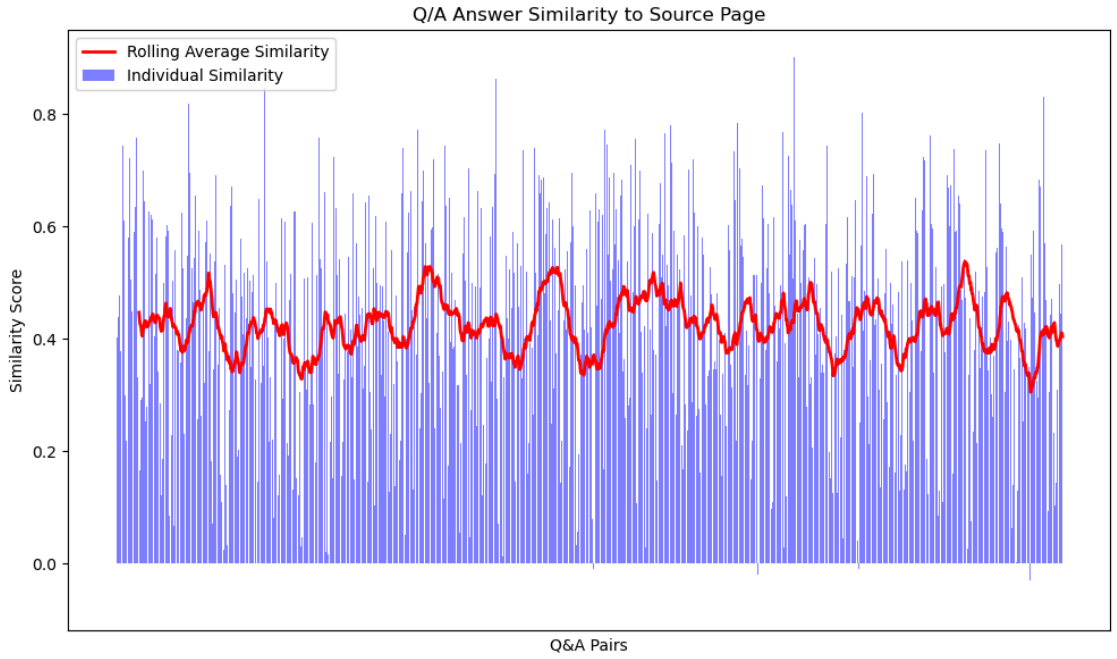

Figure 6: Cosine Similarity for Baseline GPT-4.

The figure displays cosine similarity of the answers given by the GPT4o and its truth source (the document the question came from). The graph represents cosine similarity scores for over 2000+ Q&A pairs. Individual score for each question in blue show considerable variability and the rolling average (red) of these scores seem to be stable around 0.3 to 0.5

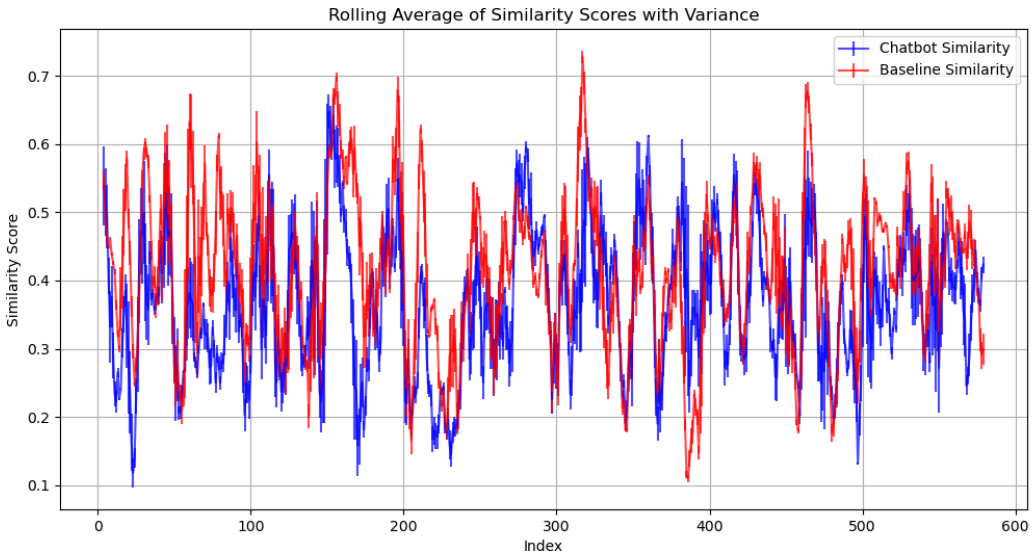

Figure 7: Comparing Rolling Averages in Cosine Similarity.

Here, we directly compare the rolling averages of cosine similarity scores between the RAG chatbot (blue) and GPT4o (red) for the same questions. We see our chatbot hold up with state-of-the art results, highlighting the capabilities the RAG and open-sourced tools have in providing reliable, contextualized results.

To summarize, our RAG chatbot holds up with current state-of-the-art LLMs in providing contextualized answers. The integration of retrieval mechanisms makes this system capable of being trusted in providing relevant answers.

6. Deliverables

- **Structured Dataset**:
  - A comprehensive dataset containing the scraped information from the targeted wiki page.
  - Organized in a tabular format (e.g., CSV or JSON) to facilitate further analysis and utilization.
- **Test Dataset (Q/A Subset)**:
  - A JSON file containing a subset of the scraped data for testing purposes.
  - Each entry includes:
    - A human-generated question.
    - A reference answer from the dataset.
    - Ranked top k relevant documents for each question.
- **Qdrant Vector Database**:
  - A vectorized representation of the scraped data, enabling efficient similarity searches for question answering.
  - Hosted locally or in the cloud, depending on deployment needs.
- **Backend Codebase**:
  - Python scripts that:
    - Scrape and process the wiki page data.
    - Populate the Qdrant vector database.
    - Automate the testing workflow by matching questions with relevant documents and reference answers.
- **Frontend Codebase**:
  - A Streamlit application to:
    - Query the Qdrant database.
    - Display the results, including ranked documents and answers.
    - Provide an interactive interface for testing.
- **Code Documentation**:
  - In-code comments explaining the purpose and functionality of each module and function.
  - A README file describing:
    - Installation steps.
    - Usage instructions for the backend and frontend components.
    - Environment setup details.
- **Distribution Documentation**:
  - Instructions for deploying the backend and frontend components on a local system or a cloud-based Nexus cluster.

7. Citations

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. Advances in Neural Information Processing Systems, 33, 9459-9474.

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., ... & Wang, H. (2023). Retrieval-augmented generation for large language models: A survey. arXiv preprint arXiv:2312.10997.

Ke, Y., Jin, L., Elangovan, K., Abdullah, H. R., Liu, N., Sia, A. T. H., ... & Ting, D. S. W. (2024). Development and Testing of Retrieval Augmented Generation in Large Language Models--A Case Study
Report. arXiv preprint arXiv:2402.01733.

Kulkarni, Mandar, Praveen Tangarajan, Kyung Kim, & Anusua Trivedi. (2024). Reinforcement Learning for Optimizing RAG for Domain Chatbots. arXiv preprint arXiv:2401.06800.

Rangan, Keshav, and Yiqiao Yin. (2024). A Fine-Tuning Enhanced RAG System with Quantized Influence Measure as AI Judge. arXiv preprint arXiv:2402.17081.

Vulić, Ivan, Pei-Hao Su, Sam Coope, Daniela Gerz, Paweł Budzianowski, Iñigo Casanueva, Nikola Mrkšić, and Tsung-Hsien Wen. (2021). ConvFiT: Conversational Fine-Tuning of Pretrained Language Models. arXiv preprint arXiv:2109.10126.

Zheng, Y., Zhang, R., Zhang, J., Ye, Y., & Luo, Z. (2024). Llamafactory: Unified efficient fine-tuning of 100+ language models. arXiv preprint arXiv:2403.13372.