Wikipedia-Savvy-RAG: Wikipedia-backed RAG system expert about STEM subjects

Dario Filatrella and **Michele Palma** and **Mattia Scardecchia** and **Federico Zarantonello**

Abstract

We present Wikipedia-Savvy-RAG, a lightweight Retrieval-Augmented Generation (RAG) system for STEM domain question answering. To improve performance in knowledge intensive tasks we filtered, cleaned, chunked and embedded the English Wikipedia and built a vector database of STEM-focused semantic embeddings. Then, we used QLORA to efficiently finetune a small pretrained language model (500M params) on a subset of the Yahoo Answers dataset, feeding the model with the most relevant passages from the vector database and keeping the query and passage embedders frozen. Finally, we used the MMLU dataset to benchmark our system and run a variety of ablation studies, using the pretrained llm without RAG as our baseline. We show that even a small LLM can benefit from retrieval, and that, surprisingly, in this regime showing more solved examples and using a more sophisticated inference strategy are both detrimental to performance. The system is lightweight and can be deployed on most modern personal computers. To demonstrate its capabilities, we developed a Streamlit web application that allows users to interact with the model in real-time with a chat-like interface. Experiments are fully reproducible and the code is available at `https://github.com/spring-projects-2024/wiki-savvy-rag`.

1 Introduction and Related Works

In recent years, the field of natural language processing has seen a surge in the development of large language models (LLMs) that can generate human-sounding text, with applications to chatbots, question-answering, text summarization and many more. (Brown et al., 2020) However, LLMs often struggle with generating accurate and informative responses, especially in specialized domains like science, technology, engineering, and mathematics (STEM). (Huang et al., 2023)

A successful approach to address this limitation has been the combination of LLMs with retrieval mechanisms from a knowledge base to generate more accurate and informative responses. Along this line of research, started with (Lewis et al., 2021), several variants and improvements have been proposed, such as (Shi et al., 2023) (Lin et al., 2024). These are extensively reviewed in (Gao et al., 2024).

In this paper, we present Wikipedia-Savvy-RAG, a lightweight RAG system fine-tuned for STEM question answering in chat format. We strived to develop a system that can be trained and deployed on cheap hardware. Indeed, our system can be deployed on personal computers with a CUDA-enabled GPU with at least 8GB of memory, through a Streamlit web application we developed, and the finetuning can be replicated on a single consumer-grade GPU in a few hours. The main research question that we aimed to answer is whether and to what extent a small LLM can benefit from retrieval in knowledge intensive tasks, and whether a simple finetuning recipe under tight computational constraints can still improve performance.

To build the vector database, we processed the English Wikipedia available on WikiMedia (wik, a) and created a vector database using the Faiss library (Douze et al., 2024) and SQLite (Hipp, 2020) to index and retrieve embeddings and passages, respectively. To keep resources low, we employed a small pretrained LLM with 500M parameters (Bai et al., 2023), and we used QLORA (Dettmers et al., 2023) to finetune it efficiently, keeping the embedder frozen, on a subset of the Yahoo Answers dataset (yah). In order to benchmark our system and conduct a variety of ablation studies, we used the MMLU dataset (Hendrycks et al., 2021).

This report will outline the steps we took to build our system, the motivations behind our choices, and the results of our evaluation. We will also discuss potential improvements and future work.

2 Knowledge Base

To build our knowledge based we downloaded and processed the English Wikipedia dump, keeping only STEM articles.

2.1 Filtering and Cleaning

We downloaded the dump dated 20 December 2023 (100GB) from WikiMedia (wik, a). Wikipedia articles have category tags to help classify them. Categories are organized in a directed acyclic graph (DAG), but not a tree (wik, b). To keep STEM articles, we marked only the categories that are at distance at most k from a set of hand-picked root categories in the category DAG, and we kept articles belonging to the marked categories. Then, we used regular expressions and many Python scripts to remove (almost) everything that wasn't plain text, such as XML tags, tables, and references. After filtering, the XML dump was reduced to 22GB. After cleaning, it went down to 8GB.

Then we proceeded in splitting the articles into the different sections to obtain the chunks that will be used by the RAG model. In the end we obtained a total of 13 million chunks, with an average length of about 230 words. We stored all of these, together with their titles, in an SQLite database.

2.2 Vector Database

We tokenized and embedded the chunks using the open-source BAAI/bge-small-en-v1.5 model (bge). Then, we used the Faiss library to build a vector database of the embeddings for efficient retrieval, using dot product to compare vectors. Faiss is a library for efficient similarity search and clustering of dense vectors that offers different indexing methods. We mostly investigated three index variants: Flat uses simple brute-force search, IVF uses centroid-based clustering to implement a hierarchical search that reduces the number of comparisons, and HNSW creates a multi-layered graph structure where each layer is a simplified, navigable small world network. For further details, see the excellent Faiss documentation (Douze et al., 2024).

Faiss also offers different quantization techniques to reduce the memory footprint of the embeddings. We focused on two quantization techniques: scalar quantization and product quantization. Scalar quantization quantizes each dimension of the vector independently in a linear range, while the product quantization splits the vector into subvectors and quantizes each subvector independently. See the Benchmark section for a comparison between these strategies.

3 Retrieval-Augmented Generation

3.1 Large Language Model

As LLM, we chose the pretrained Qwen/Qwen1.5-0.5B-Chat (qwe) model, the smallest model in the Qwen v1.5 family with only 620M parameters. It was pretrained with a large amount of data, and post-trained with both supervised finetuning and direct preference optimization to align it with human preferences. It has a context length of 32K tokens.

3.2 Generation

To generate text through RAG, we need to pass retrieved passages to the model in order to produce logits for the next token, and then use a decoding strategy to generate the final answer. We implemented two methods to produce logits. The first one, which we refer to as 'naive', concatenates all the retrieved passages and the query, and feeds the result to the model. The second one, following (Shi et al., 2023), feeds the query and each individual passage separately to the model, and then combines the predicted probabilities from each passage with weights proportional to the similarity to get probabilities. As for the decoding strategy, we implemented greedy decoding, top-k sampling, and top-p sampling. (Shi et al., 2024)

4 Finetuning

The LLM was pretrained aligned, but it was never thought to compose its answers to user queries using retrieved passages. To address this, we finetuned our system on a question answering task, using a simple recipe.

4.1 Data

As training data, we downloaded the Yahoo Answers dataset and used its metadata to filter it and keep only STEM questions. The result is around 23k examples. As validation data, we used the validation split of the MMLU dataset (mml). During training, for a given question, we retrieved the single most similar passage from our knowledge base and formatted the input to meet the expectations of Qwen, including the retrieved passage as context.

4.2 Training

We froze the query and passage embedders and finetuned the LLM only. We used the QLORA

technique (Dettmers et al., 2023) to drastically reduce the number of training parameters and the memory requirements while minimally impacting performance. In essence, this consists in freezing and quantizing the pretrained weights and adding low rank adapters to each layer, as in (Hu et al., 2021). Then, we backpropagate through the 4-bit quantized frozen weights to train the adapters. As a training criterion, we use cross entropy loss between predicted logits and the ground truth answer. This allows exploiting the inherent parallelism of the transformer architecture, since thanks to the causal mask we obtain an error signal for each ground truth token in parallel at every step. Details about our training setup can be found in out GitHub repository.

5 Demo

To demonstrate the capabilities of our system, we developed a Streamlit web application that allows users to interact with the model in real-time with a chat-like interface. The application lets users ask questions about STEM subjects, and the model will provide answers based on the Wikipedia passages it retrieves from its knowledge base. The user can decide the number of passages to retrieve, the decoding strategy (greedy, top-k, top-p), the generation type (naive or REPLUG) and other configuration parameters. Refer to our GitHub repository for more information on the application.

5.1 Arxiv

A further functionality we implemented is the ability to upload Arxiv papers in Latex format and ask questions about them. This feature is marked as experimental, as it wasn't extensively tested.

6 Benchmark

To evaluate the performance of our system, we conducted several experiments. We compared the performance of different Faiss indexes, the accuracy of the system with and without RAG, and the accuracy of the system before and after fine-tuning.

6.1 Faiss indexes

The task was to retrieve the k most similar passages to a given query coming from the MMLU dataset. The parameters we considered were the recall, the query time, and the memory usage. The recall was computed as the intersection measure between the retrieved passages and the ground truth. The variations we considered were the index type (Flat, IVF, HNSW), the number of clusters for the IVF index, the quantization technique (product, scalar), and the amount of quantization bits.

6.2 MMLU

The Massive Multitask Language Understanding (MMLU) (mml) is a collection of multiple-choice questions that spans a wide range of topics. We selected around 3000 STEM questions out of it and used them for evaluation, carrying out several ablation studies, including the number of retrieved passages, the number of examples in the prompt, and the inference strategy, on a random subsample of 1000 questions.

7 Results

7.1 Faiss indexes

We measured the recall of the indexes on k=1, 10, 50 and 100. The query time was the time it took to retrieve the 100 most similar passages for each one of the first 300 queries in the MMLU dataset with STEM subject. For the memory usage, we considered the size of the index in disk, which acts as a lower bound for the memory usage. We show in Table 1 the indexes that performed best among those we tried. More details can be found in the Appendix.

Index	Recall	Query Time	Memory
1	0.832	03:53.59	2,609
2	0.9873	03:29.50	5,219
3	0.781	01:55.76	1,740
4	0.4643	00:00.03	1,856
5	0.661	00:00.11	4,570

Table 1: Faiss index comparison. Query Time is on 300 MMLU examples and Memory is in MB. 1: Flat 4 bits scalar quantization, 2: Flat 8 bits scalar quantization, 3: Flat 128 subvectors product quantization, 4: IVF with 1000 centroids, 256 subvectors product quantization and HNSW32 as coarse quantizer, 5: HNSW 16 bits scalar quantization.

As expected, Flat indexes are the ones that perform the best in terms of recall. Here what really matters is the trade-off between compression level and accuracy. The query time significantly decreseas with the use of IVF and HNSW indexes, but the recall is also affected.

In the end, since the query time was fine for our purposes, we chose the Flat index with 128 sub-

vectors product quantization, which had the lowest memory usage and a good recall.

7.2 Training

We trained our model for slightly over 2 epochs, measuring the average cross-entropy on both the training and validation sets. We used weights and biases to monitor the training process through a variety of metrics and statistics, including parameters and gradients histograms. A selection of them is shown in the Appendix. Due to limited memory, we had to resort to gradient accumulation to simulate a larger batch size than 1 (we used 2). The training loss, with enough smoothing, decreased steadily, as expected. As for the validation loss, after an initial sharp decrease, it started increasing again. This is probably due in part to the use of different datasets for training and validation, which we did to measure generalization more reliably. The sharp drop in both losses right at the beginning of training (validation 4.4 -> 4.0, training 3.9 -> 3.5) could be due to the aggressive initial 4 bit quantization. Training curves are shown in figure 1.

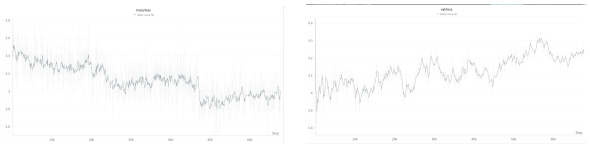

Figure 1: left: training loss curve, right: validation loss curve

7.3 MMLU

From our ablations, several findings emerged. First, pretrained Qwen can't achieve competitive performance on a challenging benchmark like MMLU. In fact, its accuracy is close to random guessing, especially without RAG. Second, even such a small model can benefit from retrieval, as shown by the significant improvement in accuracy when using RAG compared to the pretrained LLM baseline. Furthermore, averaging probabilities as in (Shi et al., 2023), the performance of the system improves with the number of retrieved passages, reaching a peak at k=3 followed by a plateau. These findings are summarized in figure 2. Third, the k-shot setting, where the model is given k examples in the prompt, is detrimental to performance (see Appendix). Probably, the increased complexity of the prompt confuses such a small model more than the demonstration helps. Fourth, we compared the naive and REPLUG inference strategies, considering the number of retrieved passages at which REPLUG peaks in performance, and we found, somewhat surprisingly, that the naive strategy outperforms the REPLUG one (0.29 vs 0.28 accuracy, see Appendix), at least with such a small model. Finally, as you can see in the training curves in 1, after an initial decrease that is attributable to recovering from 4bit quantization, the model's validation performance (on MMLU) starts increasing, suggesting that the model is not benefitting much from the finetuning.

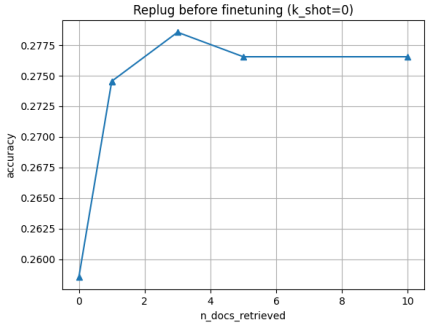

Figure 2: accuracy vs number of retrieved passages, REPLUG inference strategy

8 Conclusion

We developed, fine-tuned, and benchmarked a lightweight Retrieval-Augmented Generation (RAG) system for STEM domain discussions, curating all aspects of the database creation, training and evaluation process. To showcase our system, we created a Streamlit web application that allows to chat with our model in real-time, using RAG to retrieve from Wikipedia as well as from provided Arxiv papers.

From our experiments, we found that even a very small LLM significantly benefit from retrieval, and that, surprisingly, with such a small model concatenating all retrieved passages is more effective than averaging next token probabilities over conditioning documents. We also found that the k-shot setting is detrimental to performance with such a small model, and that finetuning is not as effective as we hoped, probably due to the small size of the model and the complexity of the task.

References

Bge model card. https://huggingface.co/BAAI/bge-small-en-v1.5.

Mmlu dataset card. https://huggingface.co/datasets/cais/mmlu.

Qwen model card. https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat.

a. Wikimedia commons. https://commons.wikimedia.org/wiki/Main_Page.

b. Wikipedia category tree. https://www.kaggle.com/datasets/kevinlu1248/wikipedia-category-tree.

Yahoo answers dataset. https://huggingface.co/datasets/yahoo_answers_qa.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-augmented generation for large language models: A survey.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding.

Richard D Hipp. 2020. SQLite.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2023. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-augmented generation for knowledge-intensive nlp tasks.

Xi Victoria Lin, Xilun Chen, Mingda Chen, Weijia Shi, Maria Lomeli, Rich James, Pedro Rodriguez, Jacob Kahn, Gergely Szilvasy, Mike Lewis, Luke Zettlemoyer, and Scott Yih. 2024. Ra-dit: Retrieval-augmented dual instruction tuning.

Beijing Academy of Artificial Intelligence. Baai - bge-large-en. https://huggingface.co/BAAI/bge-large-en.

Chufan Shi, Haoran Yang, Deng Cai, Zhisong Zhang, Yifan Wang, Yujiu Yang, and Wai Lam. 2024. A thorough examination of decoding methods in the era of llms.

Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen tau Yih. 2023. Replug: Retrieval-augmented black-box language models.

A Appendix

A.1 Faiss indexes

Index	Rank 1	Rank 10	Rank 50	Rank 100	Elapsed Time	Size on Disk (bytes)
SQ4	0.72	0.832	0.8477	0.8558	03:53.59	2,609,777,937
SQ8	0.99	0.9873	0.9885	0.9892	03:29.50	5,219,552,721
PQ64	0.4167	0.535	0.5798	0.5919	00:52.62	870,318,230
PQ128	0.63	0.781	0.8038	0.8144	01:55.76	1,740,243,158
OPQ64_256,IVF1000_HNSW32,PQ64x4fsr	0.18	0.2613	0.2797	0.2834	00:00.11	545,916,084
OPQ64_256,IVF2000_HNSW32,PQ64x4fsr	0.14	0.2493	0.2694	0.2755	00:00.01	547,695,892
OPQ64_256,IVF5000_HNSW32,PQ64x4fsr	0.1433	0.247	0.2717	0.2714	00:00.01	553,176,884
OPQ128_512,IVF1000_HNSW32,PQ128x4fsr	0.2867	0.368	0.3849	0.3829	00:00.01	982,815,924
OPQ128_512,IVF2000_HNSW32,PQ128x4fsr	0.2967	0.3433	0.3653	0.3656	00:00.01	986,153,236
OPQ128_512,IVF5000_HNSW32,PQ128x4fsr	0.25	0.3233	0.3409	0.3340	00:00.01	996,253,492
OPQ256_1024,IVF1000_HNSW32,PQ256x4fsr	0.4067	0.4643	0.4737	0.4730	00:00.03	1,856,656,564
OPQ256_1024,IVF2000_HNSW32,PQ256x4fsr	0.3833	0.438	0.4435	0.4390	00:00.02	1,862,908,180
OPQ256_1024,IVF5000_HNSW32,PQ256x4fsr	0.3467	0.388	0.3802	0.3666	00:00.02	1,882,214,196
HNSW16_SQ4	0.5933	0.661	0.5829	0.4872	00:00.11	4,570,780,394
HNSW16_PQ12	0.0	0.0037	0.0055	0.0050	00:00.01	2,124,506,683
HNSW16_PQ24	0.0	0.002	0.0021	0.0021	00:00.01	2,287,617,607
HNSW32_PQ12	0.0333	0.0733	0.1001	0.1097	00:00.02	3,862,425,891
HNSW32_PQ24	0.24	0.3193	0.353	0.3486	00:00.04	4,025,536,815

Figure 3: Extensive comparison of Faiss indexes. Rank n is the top-n recall using exhaustive search as ground truth.

A.2 Training plots

A.3 Other ablations on MMLU

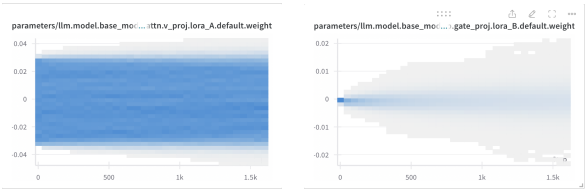

Figure 4: Evolution of the histogram of the parameters during training. left: A adapter of a self attention layer. right: B adapter of a mlp layer

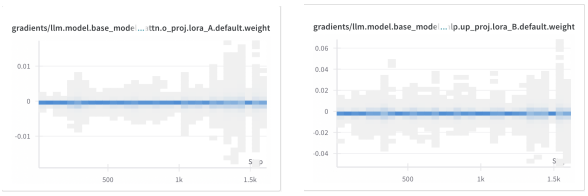

Figure 5: Evolution of the histogram of the gradients during training. left: A adapter of a self attention layer. right: B adapter of a mlp layer

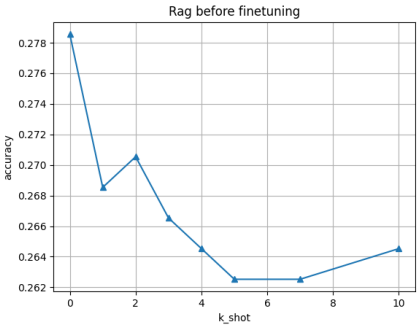

Figure 6: we compare the different number of examples (shots) given to the model in the prompt

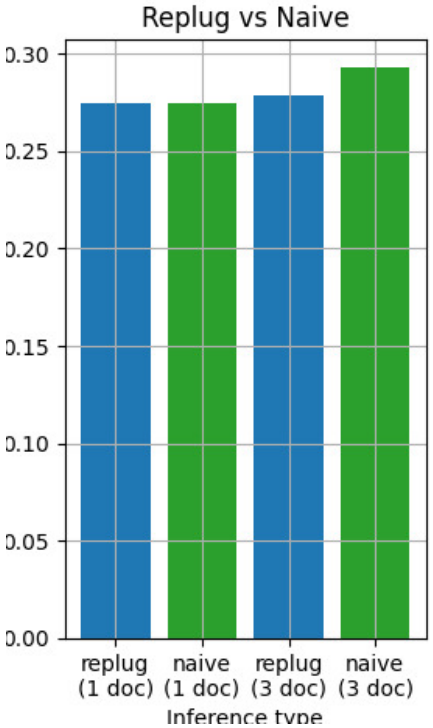

Figure 7: we compare different inference strategies (naive and replug)