Making LLMs Work for Enterprise Part 2: RAG Fine-Tuning Dataset Creation

Planning / Implementation

Large language models (LLMs) can perform useful tasks such as question answering and following instructions to generate text. Context retrieval is a popular enhancement for LLM applications: a user gives an input, the application retrieves relevant context, and the user input, along with the context are passed to the LLM for informed output generation. Retrieval augmented generation (RAG) improves the accuracy, diversity, and specificity of LLM generated responses by retrieving relevant context and adding it to the prompt (Lewis et al., 2021). However, for enterprise applications, using RAG with an open source LLM may not suffice. Using open source, instruction-fine-tuned LLMs, such as the Llama 2 Chat family, cedes the ability to tune the model output to the specific use case. Depending on the use case, enterprise teams developing RAG applications may find that open-source model responses are too verbose, prone to hallucination, or do not have the desired tone or level of detail. Fine-tuning an LLM for the specific RAG application can resolve these issues (Zhang et al., 2024).

In many cases, plenty of company-specific data–for example, existing documents or webpages–is available to serve as context in a RAG application. But the questions used for fine-tuning should be the types of questions expected from the end users of the application, and the answers should demonstrate the level of detail, tone, length, and diversity with which the application should respond.

In this article, we will discuss the steps for creating a dataset for fine-tuning a LLM to be used in a RAG application that answers questions about a company-specific knowledge base. It is unlikely that hundreds or thousands of context-question-answer examples that align with the needs of the LLM application are readily available. However, using a small set of high-quality human-created examples, we can use a high-performing LLM to generate questions and answers that relate to the available context documents. This process will augment the dataset while maintaining the question-and-answer patterns of the original human-created examples. The size of the dataset can quickly scale to thousands of examples.

The deliverables created by following the steps in this article are:

- User input scope definition
- Collection of cleaned document chunks relevant to scope
- 50 randomly selected document chunks, each with a manually created question-and-answer pair
- Several thousand LLM-generated (plus 20 manually created) document chunk-question-answer sets
- Combined dataset of positive and negative RAG question and answer examples

Requirements

The high-level requirements are as follows:

- Hardware: Server with at least 1x A100 80GB
- Software: NVIDIA Triton Inference Server

Process overview

The process is as follows:

1. Define Scope
2. Collect, Clean, and Chunk Text Documents
3. Manually Create Examples
4. Generate a Large Q&A Dataset with Few-Shot Prompting
5. Create a Positive RAG Dataset
6. Create a Negative RAG Dataset
7. Combine Positive and Negative RAG Datasets

1. Define Scope

In a perfect RAG application, a user inputs an instruction, and a system retrieves the most relevant document chunk available to help complete the task. The document chunk and question are inserted into a prompt template, and that prompt is then passed to an LLM, which outputs an accurate response to the instruction, using information from the context chunk.

First, the scope of user questions that the application should be able to respond to needs to be defined. Defining the scope as consisting of certain question types–for example, “ask for product specifications,” “ask for product descriptions,” and “ask for product recommendations”–may be helpful. It is also important to consider the specificity and complexity of user questions in the scope definition. While manually creating this small dataset, make sure the questions cover the entire scope you have defined for your application.

Deliverable: user input scope definition

2. Collect, Clean, and Chunk Text Documents

A fundamental piece of a RAG application is the document collection. Given a user input, relevant document chunks will be retrieved from the collection and passed, along with the input, to the LLM. To create this document collection, gather the documents that contain the information necessary to answer user questions within the application scope.

The document text should be parsed and cleaned. These steps depend on the document file types and the format of the text. However, in most cases, removing consecutive whitespaces, non-ascii characters, and boilerplate such as headers and footers is helpful. Also consider how text elements such as tables in your documents will be structured - converting tables to markdown can improve readability for LLMs.

Finally, break the documents into small chunks, such that several can fit into the context window. Typically, a good chunk size ranges from a single sentence to a small paragraph. Consider the structure of the documents when chunking, so sentences, paragraphs, and elements like tables are not kept in-tact as much as possible. The LLM framework LangChain has a collection of text splitter classes that can help in this step.

Deliverable: collection of cleaned document chunks relevant to scope

3. Manually Create Examples

We randomly select 50 unique document chunks from the entire collection. Then, for each of those randomly selected document chunks, we write a question that fits into the scope, for which the information in the document chunk can inform the answer. (If no in-scope question can be answered by the context document, write a value such as "[NOT IN SCOPE]" for both the answer and question.) Next, we write an answer to the question, referencing the document chunk, exactly as we would desire the application to answer the end user asking the question.

The following figure shows an example using a document chunk from Lenovo Press.

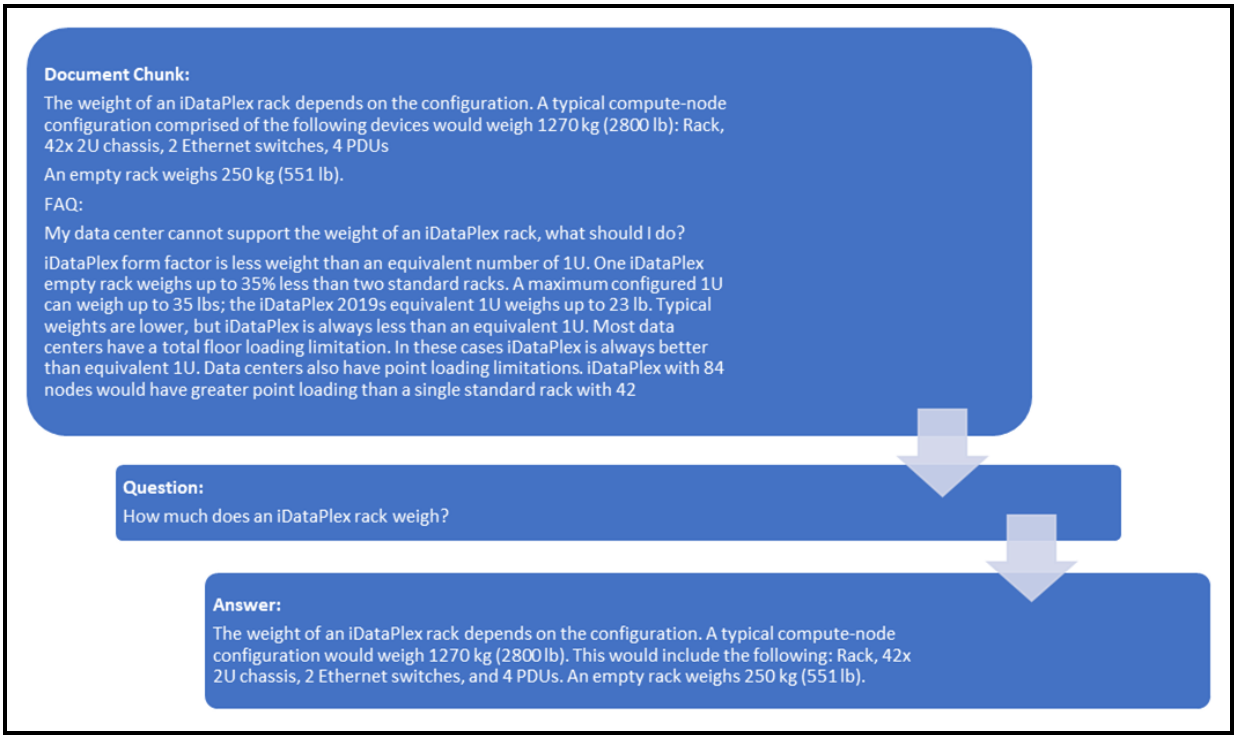

Figure 1. Document chunk from Lenovo Press

The randomly selected document chunk contains information about the weight of an iDataPlex rack. If we are building a RAG application to answer questions about technical specifications found on Lenovo Press, an in-scope question is, "How much does an iDataPlex rack weigh?" Lastly, we write the ideal response to that question, using the information found in the document chunk.

After writing question and answer pairs for 50 randomly selected document chunks, we are ready to start using an LLM to augment the dataset.

Deliverable: 50 randomly selected document chunks, each with a manually created question-and-answer pair

4. Generate a Large Q&A Dataset with Few-Shot Prompting

The LLM used for dataset augmentation should be a high performing general purpose LLM. It is important to review the LLM's license to check that it allows for commercial use and allows for its outputs to be used to train another LLM. Llama 2 13B, for example, is a high performing model that can be inferenced on a single A100 80GB. It is commercially licensed and, although in general its responses cannot be used to train another AI model, the license states that using the outputs to further train a Llama 2 family model is allowed.

Low-Rank Adaptation (LoRA) fine-tuning can yield high performing task specific LLMs with just hundreds or thousands of high-quality training examples. A high performing LLM like Llama 2 13B can capture the intent of few shot examples and generate many more examples that follow the same pattern at scale. This process can quickly augment our dataset, but the quality of data is likely to be lower than it would be if all the examples were made manually. A larger dataset can help the fine-tuned model distill the desired patterns. Therefore, it is best to yield a dataset of several thousand examples.

The strategy of few-shot prompting involves providing pairs of input and the desired model output, then appending the unlabeled input to which the model will respond. Ideally the LLM will generate an output following the pattern of the prior examples. The prompts should follow a clear format to instruct the LLM to generate the next output.

Figure 2 shows an example format for few-shot prompting the LLM to generate a question-and-answer pair for a given document chunk text.

Prompt	
Few-Shot Examples (n times)	<START> <CONTEXT> Excerpt from {file name}: {document chunk text} <QUESTION> {question} <ANSWER> {answer} <END>
Target Context	<START> <CONTEXT> Excerpt from {file name}: {document chunk text} <QUESTION>
Generated Output	
Target Question & Answer	{question} <ANSWER> {answer} <END>

Figure 2. A few-shot prompt includes n input-output pairs, followed by the target input. The model's generated response ideally will follow the pattern of the examples given in the prompt.

Details on few-shot implementation. The more examples you can provide in the prompt, the more likely the LLM will be able to generalize the pattern of your examples. Make sure at least two full examples fit into the context window. If not, the chunk size should be decreased. If *n* examples fit into your prompt, *n* unique examples should be randomly selected from the manually created dataset and added to the prompt in random order. Random selection and ordering will help the augmented dataset generalize the patterns of the manually created dataset.

Deliverable: several thousand LLM-generated (plus 20 manually created) document chunk-question-answer sets

5. Create a Positive RAG Dataset

The first major component of a RAG Q&A application will retrieve document chunks that, ideally, are relevant to the user's question. These document chunks, along with the question, are combined into a prompt that is passed to the second major component of the application, the LLM inference. The LLM's job is to determine if any of the prompt's document chunks contain the answer to the question, and, if they do, use that information to generate an accurate answer.

With a large collection of document chunks and a dataset of several thousand document chunk-question-answer sets, we can train an LLM to perform this job. However, first the data must be arranged into the format matching that of the RAG application: an input (prompt) containing document chunks and a question, paired with an output (generation) answering the question using information from the chunks.

For each of the several thousand document chunk-question-answer sets previously created, we will create a RAG training example by following this process:

1. Randomly select the total number of document chunks to include in the prompt (ranging from 1 to one fewer than the maximum number of chunks that fit into the to-be-fine-tuned LLM's context window)
2. Out of the total number of document chunks, randomly select the index for the relevant document chunk. For example, if the total number of document chunks is 4, the single relevant document chunk could be the first, second, third, or fourth document chunk in the prompt; if the total number of document chunks is 1, the single relevant document chunk will be the only document chunk included in the prompt.
3. Create the RAG prompt by concatenating document chunks that are randomly selected from the entire collection of document chunks (and are therefore irrelevant to the question). Insert the single relevant document chunk into its selected position.
4. Append the question to the end of the prompt.
5. Save the example as a JSON object with the prompt as the value for the "input" key, and the answer to the question as the value for the "output" key.

Deliverable: several thousand prompts (containing a question, preceded by one relevant document chunk and possibly irrelevant document chunks) paired with answers

6. Create a Negative RAG Dataset

Sometimes, the retrieval component of a RAG application may fail to retrieve a relevant document chunk for the user's question. A common flaw in RAG applications that use chat-fine-tuned models, such as the Llama 2 chat family, is that these LLMs tend to hallucinate inaccurate answers to questions when relevant context is not provided. LLM hallucination can lead to inaccurate responses and deteriorate users' trust in the application.

Including unanswerable examples with appropriate outputs in the fine-tuning dataset can help prevent LLM hallucination, making the application more trustworthy. The positive RAG dataset created in the prior section is used to train an LLM how to extract answers to questions when relevant context is given. Now we will make a negative RAG dataset, which will be used to train an LLM how to respond when relevant information is not available for the given question.

First, we create a collection of many different responses the LLM could give when it does not know the answer. We can manually create the first several and save them to a text file, separated by newlines. Here is an example:

```
I'm sorry, I wasn't able to find relevant information to answer that questio
n.
Sorry, I don't know the answer to that. Try asking another question.
I tried my best, but I wasn't able to find any information about that. Sorry
!
I wasn't able to find any information for your question. Maybe try rephrasin
g it.
Sorry! I don't know the answer to that. Do you want to try another question?
```

By randomly selecting and sorting these manually created examples, we can create prompts to pass to a LLM. The LLM should follow the pattern and generate new lines that follow the same pattern. Keep generating new responses that follow this pattern until there are several hundred unique responses. Manually check the resulting collection to be sure the generated responses properly follow the pattern as desired.

The next step in making a negative RAG dataset is similar to the positive RAG dataset creation process, with some important differences.

For each of the several thousand document chunk-question-answer sets previously created, we will create a RAG negative training example by following this process:

1. Randomly select the total number of document chunks to include in the prompt (ranging from 1 to one fewer than the maximum number of chunks that fit into the to-be-fine-tuned LLM's context window)
2. Create the RAG prompt by concatenating document chunks that are randomly selected from the entire collection of document chunks (and are therefore irrelevant to the question). Only use randomly selected, irrelevant document chunks; do not use the relevant document chunk in the prompt.
3. Append the question to the end of the prompt.
4. Save the example as a JSON object with the prompt as the value for the "input" key. Randomly select a negative response from the generated collection and save it as the value for the "output" key in the JSON object.

Deliverable: Several thousand prompts (containing a question, one-to-multiple irrelevant document chunks) paired with negative (i.e., "I don't know," and similar) responses

7. Combine Positive and Negative RAG Datasets

The final step in generating a dataset for RAG Q&A fine-tuning is to combine the positive and negative datasets. The ratio of positive-to-negative examples in the training dataset should be high, since positive examples contain the important pattern of extracting an answer from the provided context. Therefore, we use all the positive examples in the final dataset, but only use enough negative examples so they make up ten percent of the total dataset. If LLM hallucination is too common after the initial fine-tune, we can further fine-tune the LLM with a larger percentage of negative examples.

Deliverable: Combined dataset of positive and negative RAG question and answer examples

Conclusion

By following this process, we have defined the scope of user inputs the application should handle and created a dataset that can be used to fine-tune an LLM for the application. This dataset is designed to help the fine-tuned LLM (1) generate the desired response to in-scope questions using relevant context document chunks and (2) generate a negative response when no relevant context is provided. These features give the enterprise greater control over the LLM responses while improving the trustworthiness and accuracy of the RAG application.

Read the next article in this series : Part 3: Generative LLM Fine-tuning for RAG

For more information on Lenovo offerings for Generative AI, see the Reference Architecture for Generative AI Based on Large Language Models (LLMs), available from https://lenovopress.lenovo.com/lp1798-reference-architecture-for-generative-ai-based-on-large-language-models.

Authors

David Ellison is the Chief Data Scientist for Lenovo ISG. Through Lenovo's US and European AI Discover Centers, he leads a team that uses cutting-edge AI techniques to deliver solutions for external customers while internally supporting the overall AI strategy for the Worldwide Infrastructure Solutions Group. Before joining Lenovo, he ran an international scientific analysis and equipment company and worked as a Data Scientist for the US Postal Service. Previous to that, he received a PhD in Biomedical Engineering from Johns Hopkins University. He has numerous publications in top tier journals including two in the Proceedings of the National Academy of the Sciences.

Chris Van Buren is a Staff Data Scientist at Lenovo. He researches generative AI for enterprise use cases and has developed retrieval augmented generation (RAG) applications with open source, on-premises LLMs.

Related product families

Product families related to this document are the following:

- AI Servers
- Artificial Intelligence
- ThinkSystem SR675 V3 Server
- ThinkSystem SR680a V3 Server
- ThinkSystem SR685a V3 Server
- ThinkSystem SR780a V3 Server

Notices

Lenovo may not offer the products, services, or features discussed in this document in all countries. Consult your local Lenovo representative for information on the products and services currently available in your area. Any reference to a Lenovo product, program, or service is not intended to state or imply that only that Lenovo product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any Lenovo intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any other product, program, or service. Lenovo may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

Lenovo (United States), Inc.
8001 Development Drive
Morrisville, NC 27560
U.S.A.
Attention: Lenovo Director of Licensing

LENOVO PROVIDES THIS PUBLICATION ”AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. Lenovo may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

The products described in this document are not intended for use in implantation or other life support applications where malfunction may result in injury or death to persons. The information contained in this document does not affect or change Lenovo product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of Lenovo or third parties. All information contained in this document was obtained in specific environments and is presented as an illustration. The result obtained in other operating environments may vary. Lenovo may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any references in this publication to non-Lenovo Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this Lenovo product, and use of those Web sites is at your own risk. Any performance data contained herein was determined in a controlled environment. Therefore, the result obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

This document, LP1954, was created or updated on May 2, 2024.

Send us your comments in one of the following ways:

- Use the online Contact us review form found at:
  https://lenovopress.lenovo.com/LP1954
- Send your comments in an e-mail to:
  comments@lenovopress.com

This document is available online at https://lenovopress.lenovo.com/LP1954.

Trademarks

Lenovo and the Lenovo logo are trademarks or registered trademarks of Lenovo in the United States, other countries, or both. A current list of Lenovo trademarks is available on the Web at https://www.lenovo.com/us/en/legal/copytrade/.

The following terms are trademarks of Lenovo in the United States, other countries, or both:
Lenovo®

Other company, product, or service names may be trademarks or service marks of others.